



City Research Online

City St George's, University of London

Citation: Booth, T. (2022). Tactical troubleshooting: investigating support for end-user developers in physical computing. (Unpublished Doctoral thesis, City, University of London)

This is the accepted version of the paper.

This version of the publication may differ from the final published version. To cite this item please consult the publisher's version.

Permanent repository link: <https://openaccess.city.ac.uk/id/eprint/28403/>

Copyright and Reuse: Copyright and Moral Rights remain with the author(s) and/or copyright holders. Copies of full items can be used for personal research or study, educational, or not-for-profit purposes without prior permission or charge, unless otherwise indicated, provided that the authors, title and full bibliographic details are credited, a hyperlink and/or URL is given for the original metadata page and the content is not changed in any way. For full details of reuse please refer to [City Research Online policy](#).

Tactical Troubleshooting:

Investigating Support for End-User Developers in Physical Computing

Tracey Booth

A dissertation submitted in partial fulfilment
of the requirements for the degree of

Doctor of Philosophy

at

City, University of London

Centre for Human-Computer Interaction Design
Department of Computer Science
School of Mathematics, Computer Science, and Engineering

April 2022

Supervisors: Dr Simone Stumpf
Dr Sara Jones
Dr Jon Bird

Copyright statement

Powers of discretion are hereby granted to the Librarian of City, University of London, to copy the thesis in whole or in part without further reference to the author. This permission covers only single copies made for study purposes, subject to normal conditions of acknowledgement.

Abstract

With maker culture and its emphasis on DIY and personal creation now firmly embedded in society, more end-user developers—for example, artists, hobbyists, researchers and designers—have been drawn towards developing interactive physical computing devices—microcontroller-based systems that interact with the physical environment via sensors and actuators—using popular development platforms such as Arduino. However, developing these devices usually involves 1) constructing an electronic circuit and 2) programming its behaviour—activities which can present challenges to end-user developers, particularly inexperienced ones. Inability to overcome these challenges may result in them failing to complete their projects or even abandoning their physical computing ambitions altogether.

Decades of research have focused on understanding the problems that end-user programmers and novice programmers face when programming software, and developing support for overcoming these. Physical computing development might benefit from similar approaches but prior to this thesis there had been little empirical work to determine what difficulties end-user developers experience in this domain, their natural behaviours when dealing with them, or how they might be supported in overcoming them.

To fill this gap, this thesis aims to answer the following, overarching research question:

How can end-user developers be supported in overcoming problems they experience when developing physical computing artefacts?

Answering this question involved four stages of work:

1. An exploratory, empirical study, investigating the problems that end-user developers experience when developing a physical computing device.
2. Deeper analysis of data from the same study, to identify end-user developers' natural behaviours when troubleshooting *circuit* bugs—the type of bug identified as most likely to impede success.
3. Informed by this empirical work and inspired by creativity support cards, the development of a novel, physical card-based tool, to support end-user developers when troubleshooting.
4. Evaluation of the support tool in a study with novice end-user developers, to observe its impact on troubleshooting, and elicit feedback about their first-hand experience of using it.

This research is the first empirical investigation into the problems, behaviours and support needs of adult, end-user developers using platforms such as Arduino to develop physical computing devices. The main contributions of this thesis, are:

1. Empirically grounded knowledge of the problems experienced by end-user developers when constructing and programming a physical computing device, including the types, location and number of problems, and whether they are overcome.
2. Empirically grounded knowledge of end-user developers' natural behaviours when troubleshooting circuit bugs that they have introduced during development, including the tactics they employ, resulting in suggestions for types of support from which end-user developers might benefit when troubleshooting in this domain.
3. A novel, physical card-based tool to support end-user developers in troubleshooting physical computing problems. The tool provides ideas for different troubleshooting tactics and is designed to encourage more thoughtful troubleshooting.
4. Insights into how a physical card-based support tool might be used and perceived by novice end-user developers when troubleshooting circuit bugs in a physical computing task.

Acknowledgements

There are people without whom this PhD would not have been possible. Chief among them (and I know he'll enjoy that accolade), is my husband, Olivier, who made many sacrifices and adjustments to facilitate and support my doctoral adventure, was patient (largely) in awaiting its completion, and never once doubted that I would succeed, even during my wobbliest wobbles. Love and gratitude, always.

My PhD supervisors—Dr Simone Stumpf, Dr Sara Jones and Dr Jon Bird—were exceptional in the extent of their engagement with my work and their efforts on my behalf. Generous with their advice and assistance, and a constant source of support and encouragement, I really did land with my bum in the supervisory butter. Heartfelt thanks to all. Special long service recognition to Simone, who was along for the entire ride, starting with supervision of the MSc dissertation project that spawned the topic of my PhD.

I would also like to express my gratitude to my examiners, Professor Judith Good and Professor Stephanie Wilson, for their thorough critical review of my work, fair examination, and constructive feedback, from which my thesis has benefitted greatly. Much appreciation to Dr Ernesto Priego, for chairing my viva so beautifully, but also for providing help and support during my PhD, as Senior Tutor for Research.

Completing the final stages of this PhD during a global pandemic was not ideal. To my fellow PhD students—particularly Carol, Beatrice, Niamh, Alex and Axel—thank you for the online meetups that helped to re-establish some working routine in a time of upheaval and great uncertainty. Congratulations on your own achievements. Whatever you accomplished during this period, you should be proud.

As a member of HCID, I have had the pleasure of working and studying alongside many remarkable humans. Thank you to all, for any part played in my PhD, or my evolution as a researcher, but also for the company and friendship within and outside of the university walls, and for setting the bar for 'work' parties high. Particular thanks to Stuart, for being so flexible and accommodating with access to the Interaction Lab for my studies, and to Steph, for numerous reasons, but not least the conversation, over a beer, several years ago, that made me think that doing a PhD might *actually* be possible.

Many thanks to City, University of London for waiving student fees for the duration of my PhD, and to all course officers—Ann Marie, David, Nathalie, Savita—who provided behind-the-scenes help and support.

I am indebted to all of my study participants, for giving so generously of their time, rising to the challenge of the tasks I set, and being candid in their feedback. I am also grateful to everyone who helped me to recruit participants, as well as to anyone in the wider HCI research community who provided me with feedback, inspiration, challenge, help or encouragement.

Some fabulous friends played a large part in keeping me, variously, sane, entertained, motivated and on track, offered their support, or assisted me in some way. Special acknowledgement to—in alphabetical order—Adrian, Bernard, Carol, Emma, Gita (RIP), Leon, Manda, Monty, Moose, Nathan, Niamh and Simon.

Finally, my academic journey has been somewhat unconventional, but my family have been supportive throughout. Much love and gratitude to my parents, for their unwavering faith in my abilities, and for preparing me for this endeavour by raising me to be curious, questioning, and tenacious in pursuing my interests. I hope that this new achievement makes up for me still not completing a bachelor's degree. Love and appreciation also to my sister Vicky, for every instance of cheerleading, and to my niece, Olive, for recreational distraction from stuff that wasn't quite as much fun as Minecraft.

Table of Contents

List of Figures	i
List of Tables	v
Publications arising from this thesis.....	vi

Chapter 1 Introduction 1

1.1 Background and motivation	1
1.1.1 The Maker Movement	1
1.1.2 Physical computing.....	2
1.1.3 End-user development.....	2
1.1.4 End-user developers in physical computing: <i>Makers</i>	3
1.1.5 Supporting end-user developers' physical computing development.....	4
1.1.6 HCI research in the physical computing domain	6
1.1.7 A physical card-based tool to support end-user developers' troubleshooting.....	7
1.2 Research questions	8
1.3 Summary of contributions.....	9
1.4 Research Scope	10
1.5 Methodology and approach.....	11
1.5.1 An empirical, user-centred approach.....	11
1.5.2 Research methods and methodological stance.....	13
1.6 Thesis outline	17

Chapter 2 Related Work 19

2.1 End-user developers	20
2.1.1 End-user vs novice and professional programmers	20
2.1.2 The disadvantages of non-experts.....	22
2.1.3 End-user developers as <i>makers</i>	22
2.2 Non-experts' problems in programming.....	23
2.2.1 Causes of software error.....	24
2.2.2 Learning barriers	25
2.2.3 Strategies.....	27
2.3 Non-experts' problems with circuits	29

2.3.1	Circuit theory: problematic for learners.....	29
2.4	Problems affecting end-user developers in physical computing.....	31
2.4.1	Existing evidence of problems in physical computing.....	32
2.5	Supporting end-user developers.....	34
2.5.1	The challenge of supporting end-user developers	34
2.6	Supporting non-expert programmers.....	35
2.6.1	Making programming easier	36
2.7	Supporting physical computing development	37
2.7.1	Easier programming in physical computing.....	37
2.7.2	Easier circuits in physical computing.....	38
2.7.3	Making circuits and programming easier.....	40
2.8	Supporting troubleshooting and debugging	43
2.8.1	Troubleshooting software problems (debugging)	45
2.8.2	Troubleshooting physical computing problems.....	46

Chapter 3 Problems experienced by end-user developers in a physical computing task (Study 1A) 48

3.1	Introduction.....	48
3.2	Method.....	49
3.2.1	Overview	49
3.2.2	Participants	50
3.2.3	Materials.....	55
3.2.4	Procedure	61
3.2.5	Data collection	64
3.2.6	Data analysis.....	69
3.3	Results	80
3.3.1	How many problems? (RQ1).....	80
3.3.2	What types of problems? (RQ1)	80
3.3.3	Where did problems occur? (RQ1)	81
3.3.4	Did self-rated expertise and self-efficacy have an effect? (RQ2).....	83
3.3.5	Were problems overcome? (RQ3)	85
3.3.6	What went fatally wrong? (RQ3).....	87
3.4	Discussion.....	91

Chapter 4 How end-user developers troubleshoot circuit bugs (Study 1B)93

4.1	Introduction.....	93
4.2	Method: Data Analysis.....	94
4.2.1	Overview	94
4.2.2	Event Type codes.....	98
4.2.3	Runs	100
4.2.4	Episodes.....	101
4.2.5	Activity Type codes.....	102
4.2.6	Tactics codes	103
4.2.7	Bugs	106
4.3	Results	106
4.3.1	How do end-user developers troubleshoot circuit bugs? (RQ1)	107
4.3.2	Are end-user developers' troubleshooting behaviours effective? (RQ2)	114
4.4	Discussion	121
4.4.1	Troubleshooting tactics	122
4.4.2	Supporting end-user developers' troubleshooting	125

Chapter 5 Developing a physical card-based tool to support end-user developers' troubleshooting.....131

5.1	Introduction.....	131
5.2	Designing cards to support troubleshooting	132
5.2.1	Why cards?.....	133
5.2.2	Design review of existing card sets	134
5.2.3	Considerations when designing cards.....	135
5.3	Initial prototype.....	137
5.3.1	Proof of concept (informal pilot)	139
5.4	A study to inform the design of the card deck.....	140
5.4.1	Focus group sessions.....	141
5.5	The Tactical Troubleshooting toolkit	144
5.5.1	Tactic cards	146
5.5.2	Category cards.....	149
5.5.3	Best Practice cards.....	149
5.5.4	Component cards	150
5.5.5	Playmat.....	150
5.5.6	Card stand	152

5.6 Card production process.....	152
5.7 Discussion	153

Chapter 6 Evaluating the troubleshooting support cards with novice end-user developers (Study 2).....157

6.1 Introduction.....	157
6.2 Method.....	157
6.2.1 Overview	157
6.2.2 Study design	158
6.2.3 Participants.....	159
6.2.4 Materials.....	165
6.2.5 Procedure	172
6.2.6 Data collection and analysis	176
6.3 Results	180
6.3.1 What effect do the cards have on helping end-user developers troubleshoot?.....	181
6.3.2 How do end-user developers view the support tool?.....	186
6.4 Discussion	206
6.4.1 Task-related frustration.....	206
6.4.2 Summary	207

Chapter 7 Discussion and conclusion211

7.1 Contributions.....	212
7.1.1 Contribution 1	212
7.1.2 Contribution 2	214
7.1.3 Contribution 3	217
7.1.4 Contribution 4	219
7.2 Limitations of the work.....	221
7.3 The focus on troubleshooting.....	224
7.4 Reflection on methods and approach	226
7.4.1 Task observation in a laboratory environment, under tight constraints	227
7.4.2 Problems with think aloud	228
7.4.3 Approach to analysis.....	229
7.4.4 ‘What works’ vs ‘What might work better?’	229
7.5 Opportunities for future work.....	230

Appendices.....233

Appendix A.	Study 1A Ethics application.....	233
Appendix B.	Study 1A Recruitment poster	236
Appendix C.	Study 1A Participant information sheet	237
Appendix D.	Study 1A Informed consent form	238
Appendix E.	Study 1A Background Questionnaire	239
Appendix F.	Study 1A Self-efficacy questionnaire.....	242
Appendix G.	Study 1A Task instruction sheet	243
Appendix H.	Study 1B Troubleshooting flow diagram	244
Appendix I.	Initial set of candidate tactics.....	245
Appendix J.	Card design focus groups ethics application	246
Appendix K.	Card design focus groups participant information sheet.....	249
Appendix L.	List of cards used in Study 2.....	250
Appendix M.	Tactics and Best Practice cards.....	251
Appendix N.	Study 2 Ethics application	256
Appendix O.	Study 2 Recruitment flyer.....	260
Appendix P.	Study 2 Participant information sheet.....	261
Appendix Q.	Study 2 Informed consent form.....	262
Appendix R.	Study 2 Background questionnaire	263
Appendix S.	Study 2 Support Materials Questionnaire	265
Appendix T.	Study 2 Task instructions.....	267
Appendix U.	Study 2 Interview topic guide	268
Appendix V.	Study 1A/1B Participant background data	269
Appendix W.	Study 2 Participant background data	270

Bibliography.....271

List of Figures

Figure 1.	An Arduino UNO board and the Arduino IDE	5
Figure 2.	General model of troubleshooting. From Katz and Anderson, 1987	44
Figure 3.	Participants' experience, in years	53
Figure 4.	Participants' training	53
Figure 5.	Participants' self-ratings of expertise in physical computing, programming and electronics	54
Figure 6.	Individual participants' self-ratings of expertise (1-7)	54
Figure 7.	Individual participants' self-efficacy scores (out of a maximum of 100)	54
Figure 8.	Love-O-Meter prototype in action	56
Figure 9.	Model Love-O-Meter circuit	57
Figure 10.	Model Love-O-Meter program	59
Figure 11.	Study 1A: Sequence of activities	62
Figure 12.	Desk and equipment setup. An additional monitor (visible on the right-hand side of the image) mirrored the participant's screen, enabling me to observe on-screen activity during the task in an unobtrusive way.	66
Figure 13.	Still from composite split-screen video of a participant undertaking the task, showing 1) on-screen activity (large panel), 2) view of the participant's head and shoulders (small panel embedded within the screen activity panel), 3) desk-facing view (top right panel) and 4) overhead, zoomed-in view of the circuit (bottom right panel)	67
Figure 14.	Extract from a transcript spreadsheet	68
Figure 15.	Photographs of a participant's circuit	68
Figure 16.	Fritzing image of a participant's circuit, showing the Arduino board (left) and the solderless breadboard (right)	69
Figure 17.	Portion of transcript coded with problem type (Obstacle, Breakdown, Bug) and location (Code abbreviations: C=Circuit; B=Both, i.e., Circuit+Program)	78
Figure 18.	Total number of problems per participant. Participants whose columns are green <i>successfully</i> completed the task, i.e., developed a working prototype that met the task brief/specification.	80
Figure 19.	Number of each problem type, per participant, grouped by task success, ordered by total number of problems (obstacles + breakdowns + bugs)	81
Figure 20.	Number of problems by location	82
Figure 21.	Problem types by location	82
Figure 22.	Participants' self-efficacy scores (out of 100), and task success/failure	84

Figure 23.	Participants' stacked self-ratings of expertise in programming, electronics, and physical computing (each out of 7).....	84
Figure 24.	Stacked count of problems encountered by each participant, in the program, circuit and circuit+program locations.....	84
Figure 25.	Number and proportion of problems (obstacles and bugs) overcome, or not resolved.....	85
Figure 26.	Percentage of problems (bugs + obstacles) overcome, by each task success group.....	85
Figure 27.	Circuit correctness as a factor in performance in overcoming obstacles	87
Figure 28.	Hierarchy of units of analysis , and the coding schemes applied at each level. A task contains one or more troubleshooting runs. A run consists of one or more episodes. An episode consists of one or more events.	95
Figure 29.	Event codes applied to a portion of a transcript spreadsheet. A letter within a coloured (coded) cell denotes the location subcode (C=Circuit; P=Program; B=Both, i.e., Circuit+Program). Note that rather than a location sub-code, the <i>Fault recognition</i> code contains a flag ('1') to indicate the point at which the participant became aware of the problem.....	100
Figure 30.	Number of Diagnose, Fix and Evaluate Fix episodes, per participant.....	108
Figure 31.	Activity type episode counts, also indicating episodes where more than one Activity Type was coded, most notably where participants were evaluating and diagnosing (70 episodes), i.e., they were unsure if a fix had been successful.....	108
Figure 32.	Transitions between troubleshooting activity types.....	109
Figure 33.	Tactics observed in Diagnose episodes.....	111
Figure 34.	Tactics observed in Fix episodes	113
Figure 35.	Tactics observed in Evaluate Fix episodes	114
Figure 36.	Mean number of activity type episodes per circuit outcome group.....	115
Figure 37.	Mean number of bugs added/fixed per circuit outcome group	115
Figure 38.	Mean number of episodes coded with each Diagnose tactic used, by circuit outcome group.....	116
Figure 39.	Outcome of tactics employed in Fix episodes, across the whole sample.....	118
Figure 40.	Episode outcomes of the two main Fix tactics used by participants	119
Figure 41.	Mean number of episodes coded with each Fix tactic used, by circuit outcome group.....	120
Figure 42.	Mean number of episodes coded with each Evaluate Fix tactic used, by circuit outcome group.....	121
Figure 43.	Content from the initial prototype of the troubleshooting support tool	137
Figure 44.	An example card from each category	145
Figure 45.	Tactics card design, front (left) and rear (right)	147
Figure 46.	Category card design, front (left) and rear (right).....	149

Figure 47. Best Practice card design	149
Figure 48. Component card design.....	150
Figure 49. Troubleshooting toolkit playmat	151
Figure 50. Card stand	152
Figure 51. Tactics cards (front sides only).....	155
Figure 52. Category cards (fronts only), Best Practice cards, Component cards, Playmat, Card stand	156
Figure 53. Participants' training.....	162
Figure 54. Participants' length of experience (in years)	163
Figure 55. Participants' self-rated expertise	163
Figure 56. Individual participants' self-rated expertise, from 1 (Complete beginner) to 7 (Complete expert)	164
Figure 57. Participants' self-rated troubleshooting expertise in Arduino, Electronics, Programming.....	164
Figure 58. Individual participants' self-rated troubleshooting expertise: bugs in Arduino projects, circuit bugs and program bugs, from 1 (Complete beginner) to 7 (Complete expert)	164
Figure 59. Cards in card stand, and playmat.....	166
Figure 60. Buggy Prototype A (Task 1)	169
Figure 61. Buggy Prototype B (Task 2).....	170
Figure 62. Still image from the video in which the correct prototype behaviour at runtime was demonstrated, showing the temperature sensor and LEDs, but none of the wiring.....	172
Figure 63. Session sequence of activities for participants in each of the two groups	174
Figure 64. Setup for With Support task. The troubleshooting cards can be seen, in their stand, top left, and the playmat is bottom left. The buggy prototype is in a taped area, directly in front of the participant, with the parts kit above it.....	175
Figure 65. Still from a composite video of a participant task recording, showing in clockwise order from top left 1) the participant's head and shoulders view, 2) desk-facing view, 3) screen capture, 4) overhead view of the circuit and 5) over-the-shoulder view of support materials use	177
Figure 66. Photograph (left) and Fritzing image (right) capturing the final state of a prototype at the end of a task.....	178
Figure 67. Participants who achieved task success, with and without the support materials, in each task	181
Figure 68. Preseeded bugs fixed with and without the support materials in each task.....	182
Figure 69. Participants (n), per support condition, who fixed specific preseeded bugs in each task. The WSNS group had the support materials in T1; NSWNS had them in T2.	183

Figure 70.	Participants (n) per support condition, with bugs (preseeded, new, new circuit bugs, new program bugs) remaining at the end of each task. (Note: 'New bugs' counts participants with any new bugs remaining, irrespective of their location).....	184
Figure 71.	Participants (n) with bugs (pre-seeded, new bugs (all), new program bugs and new circuit bugs) remaining at the end of tasks with or without support	184
Figure 72.	Participants (n) who fixed none, some or all of the preseeded bugs, with and without support.	185
Figure 73.	Participants (n) per support condition who fixed none, some or all of the preseeded bugs in each task.....	185
Figure 74.	Support Materials questionnaire responses. Participants gave each question a rating from 1 to 7. The number within each coloured box represents the number of participants who chose that rating. Green represents the midpoint (rating = 4)	187

List of Tables

Table 1.	Study 1A inclusion/exclusion criteria for participation	50
Table 2.	Study 1A participants	52
Table 3.	Problem Type coding scheme	75
Table 4.	Example chain of problems (partial), showing Problem Type codes.....	76
Table 5.	Problem Location coding scheme, with code abbreviation in brackets beneath the code name	77
Table 6.	Example of Problem Location codes applied in conjunction with Problem Type codes.....	77
Table 7.	Participants' task performance and success	86
Table 8.	Troubleshooting event type codes	98
Table 9.	Activity Types coding scheme	102
Table 10.	Tactics coding scheme and frequency of code application (total count of episodes coded) across the sample	105
Table 11.	Summary of participants' troubleshooting of circuit bugs and the outcomes thereof	107
Table 12.	Activity Type episode counts, and the percentage of all (439) episodes these represent.....	108
Table 13.	Structure and content of the initial prototype card deck, and the relationship of the tactic categories to the support recommendations from study 1B	139
Table 14.	Card categories and their contents.....	145
Table 15.	Study 2 inclusion/exclusion criteria for participation	160
Table 16.	Study 2 Participants.....	161
Table 17.	Study 2 Participant length of experience (in years)	163
Table 18.	Study 2 Participant perceived expertise (1-7)	163
Table 19.	Study 2 Participants perceived expertise in troubleshooting (1-7).....	164
Table 20.	Participant task groups and order of conditions	173

Publications arising from this thesis

- Booth T. (2015). Investigating the Barriers Experienced by Adult End-User Developers When Physical Prototyping. In: Díaz P., Pipek V., Ardito C., Jensen C., Aedo I., Boden A. (eds) End-User Development. IS-EUD 2015. Lecture Notes in Computer Science, vol 9083. Springer, Cham. (Doctoral Consortium paper)
- Booth, T. (2015). Making Progress: Barriers to Success in End-User Developers' Physical Prototyping. 2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC), Atlanta, GA, 2015, pp. 299-300. doi: 10.1109/VLHCC.2015.7357236. (Doctoral Consortium paper and poster presentation)
- Booth, T., Stumpf, S., Bird, J. and Jones, S. (2016). Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems (CHI '16), 3485–3497. New York, NY, USA: ACM. doi:10.1145/2858036.2858533.
- Booth, T., Bird, J. and Stumpf, S. (2017). The Trouble with Troubleshooting. Presented at the CHI 2017 workshop: 'Open Design 'Intersection of Making and Manufacturing'. (Workshop paper)
- Booth T., Bird J., Stumpf S., Jones S. (2019). Designing Troubleshooting Support Cards for Novice End-User Developers of Physical Computing Prototypes. In: Malizia, A., Valtolina, S., Morch, A., Serrano, A. and Stratton A. (eds) End-User Development. IS-EUD 2019. Lecture Notes in Computer Science, vol 11553. Springer, Cham. doi.org/10.1007/978-3-030-24781-2_15

Chapter 1

Introduction

The Maker Movement and the growth of maker culture worldwide, has drawn more non-experts, for example, hobbyists, artists, designers and researchers, to physical computing development—constructing and programming interactive prototypes and devices themselves, rather than relying on professionals to do it for them (Dougherty, O'Reilly, and Conrad 2016). Popular open-source physical computing platforms such as Arduino have lowered the barriers to this type of development, however, as I will demonstrate in this thesis, this new population of end-user developers, who may lack formal training in electronics and/or programming, can still struggle to engineer and troubleshoot their creations.

This thesis seeks to understand the challenges faced by end-user developers when developing physical computing artefacts, and investigates how to support them.

The remainder of the Introduction chapter is structured as follows: I first motivate the work described in the thesis, introducing the domain and population of study, and outlining the research gaps to be addressed. I then present the research questions that were investigated in the course of this work, outline the resulting contributions, and clarify the scope of the research. Finally, I summarise the methodological stance from which the work was conducted, and the primary research methods, and conclude with an outline of the structure of the thesis.

1.1 Background and motivation

1.1.1 The Maker Movement

The emergence of the Maker Movement (Dougherty 2012), in the early 2000s, saw a cultural shift towards DIY and personal creation (Mota 2011), facilitated by advances in technology that

connected individuals and communities across the globe, and made affordable, open source software and hardware widely available. The tools of technological production are now no longer just in the hands of experts (Kuznetsov and Paulos 2010) and a wealth of online resources, underpinned by platforms and technologies that facilitate communication and sharing, provide novice makers with tutorials, guides, inspiration, encouragement and feedback. Offline maker communities have also been key to this growth. Numerous local makerspaces, hackerspaces, Fab Labs and maker groups—both formal and informal—have emerged worldwide, as people explore physical computing, digital fabrication and other types of making, in the places where they live, study and work, accessing and sharing resources, projects and expertise with other makers, in social, educational, work and community environments.

One of the most popular activities in this space is *physical computing development*, attracting many different types of established or fledgling maker, excited by the possibilities it affords, or even just for the pleasure of making something which interacts with the world.

1.1.2 Physical computing

Physical computing integrates the digital world with the physical world, usually in the form of electronic devices or systems that interact with the environment (Igoe and O’Sullivan 2004), affording opportunities for new interfaces and interaction. These artefacts often take input through sensors that measure aspects of the environment, such as temperature, proximity, or light, and respond in some way, for example, through sound, motion or vibration, using actuators (Bird, Marshall, and Rogers 2009; Gallacher et al. 2015).

Developing a physical computing artefact usually involves constructing a physical, microcontroller-based prototype (electronic circuit) and programming its behaviour. It therefore requires some measure of skill in both electronics and programming—activities which can present challenges for *end-user developers*, particularly those without training or experience in either or both of these disciplines.

1.1.3 End-user development

Put simply, *end-user development* is activity in which individuals—often non-expert developers—engage in some aspect of development, to support their own work or personal goals. In the

context of programming software, this is also referred to as *end-user programming*—a term initially popularised by Nardi (1993).

This thesis focuses on end-user development in a *physical computing* context. As physical computing incorporates constructing physical circuits (hardware), as well as writing programs (software) to control their behaviour, I have adapted¹ an existing definition of end-user development, extending it beyond software alone, to be applicable to this domain:

“End-User Development, in a physical computing context, is a set of methods, techniques, and tools that allow users of hardware and software systems, who are acting as nonprofessional developers, at some point to create, modify, or extend a physical computing artifact.”

(adapted from Lieberman et al. 2006 p.2)

End-user programming and end-user development have been studied extensively in domains such as spreadsheets (formulas are considered to be a type of programming) and web development, particularly to identify and understand the difficulties that end-user developers experience in these activities, their behaviours when programming and debugging, and to determine how they might be supported. However, until my work, little was known about end-user development in the domain of physical computing. A study that I conducted during my MSc uncovered some evidence of *learning barriers* for novice end-user developers undertaking short programming tasks in visual and textual Arduino programming environments (see section 2.2.2; full details published in Booth and Stumpf 2013), however, to my knowledge, until the work in this thesis, there had been no further investigation into the difficulties experienced by end-user developers when developing physical computing prototypes—constructing and programming microcontroller-based circuits—nor any work to understand their behaviours, or establish how they might be supported in this type of endeavour. This thesis addresses that gap.

1.1.4 End-user developers in physical computing: *Makers*

To understand end-user development in this domain, we can first look at who is doing it and to what ends. As *makers*, end-user developers’ applications of physical computing technologies are multifarious, for example:

¹ The underlined portions of text indicate the amendments made to the original definition, extending it beyond just software.

- Artists create interactive artworks that express their artistic vision (Gibb 2010).
- Designers and researchers develop devices to conduct research, explore ideas and prototype solutions to real-world problems (Cressey 2017).
- Hobbyists build smart home devices for the Internet of Things, monitoring and interacting with their home environments (Jenkins and Bogost 2014).
- Individuals with chronic conditions take charge of their own well-being and health by adapting medical devices and developing health-related information appliances (Ananthanarayan et al. 2014)(O’Kane et al. 2016).
- Hobbyist e-textile crafters develop interactive soft-circuits, embedding microcontrollers into clothing and accessories (Buechley and Hill 2010).
- Amateur and professional scientists build and adapt devices that enable them to tinker and experiment with materials, to “open source science” (Kuznetsov et al. 2012).

As new and different groups become aware of physical computing technologies and the creative and functional possibilities they afford, new applications of physical computing emerge (Buechley and Perner-Wilson 2012). However, while the backgrounds and motivations of end-user developers may vary, physical computing development requires a number of core skills that can prove challenging to those without training, irrespective of their goals.

1.1.5 Supporting end-user developers’ physical computing development

Programming is challenging for non-experts, as is *electronics engineering*. Prior to the work in this thesis it had already been suggested that in combining both of these activities, physical computing development potentially puts even greater demand on end-user developers, in terms of the knowledge and skill required to build—and troubleshoot—their creations (Tetteroo, Soute, and Markopoulos 2013).

For many years, physical computing development presented significant barriers to those without expertise in electronics and programming. Creating circuits required in-depth knowledge of electronics and electrical circuit theory; programming circuits required firstly, the ability to program in itself, and secondly, expert knowledge of how to interface software with hardware, including uploading programs to microcontrollers and controlling hardware

components programmatically. Additionally, many of the tools required for physical computing development were not easily accessible to those outside the domains of engineering and programming, due to cost, for example, and/or availability, and were often difficult to master. Thus, developing physical computing artefacts was often beyond the reach of those without specialised knowledge and/or access to specialised tools and equipment.

However, in recent years, platforms and tools have emerged that aim to make physical computing development easier for non-experts, including children and end-user developers. A key example is *Arduino* (Mellis et al. 2007)—a low-cost, open source prototyping platform, comprising a range of microcontroller boards of varying specification, and a simple, notepad-style development environment (IDE) (Figure 1). Originally developed to teach physical computing to interaction designers, the Arduino platform has achieved wide adoption by many types of end-user developer, both expert and non-expert. Championed by high profile names within the maker community (e.g., ‘Make’; n.d.), and endorsed, from its early days, as an easy point of entry into the physical computing world, the Arduino platform is the foundation upon which immeasurable numbers of physical computing projects have been developed. As the de facto physical computing platform of choice for end-user developers, it was an appropriate platform to use in the research described in this thesis, where, as I demonstrate, there is still some way to go in identifying and addressing the challenges that end-user developers face in physical computing development.



Figure 1. An Arduino UNO board and the Arduino IDE

1.1.6 HCI research in the physical computing domain

Making is empowering, and can be transformative—society’s relationship with technology is evolving as a result (Mota 2011). By engaging in DIY physical computing development practices, end users no longer just *consume* technology, they are *producing* it, making this an area of interest to HCI research, including those interested in facilitating or supporting the building of interactive systems and devices by a broader range of individuals, not just experts or those with a technical background. (De Roeck et al. 2012).

Prior to the work described in this thesis, there had, however, been little user research to determine what *difficulties* end-user developers experience when developing physical computing devices, their *behaviours* when doing so, or how to *support* them in overcoming their problems. Much of the research into physical computing had focused on 1) developing novel development tools and/or investigating the adoption or potential uses of them, or 2) assessing the opportunities and impact of these technologies, as well as wider maker culture and practices, in respect to specific groups—for example, children and young people—or society at large—for example, education, innovation, and manufacturing. Focusing more on the outputs, outcomes and opportunities of physical computing, not much attention had been paid to the challenges of actually *doing* it, particularly for adult end-user developers, who often operate outside of supported/formal learning environments.

As this thesis will demonstrate, much remains to be done to ensure that physical computing development is truly accessible to all those who are enticed to try it, or who may benefit from being able to develop or adapt a physical computing device. While platforms such as Arduino make physical computing development easier and more accessible than it used to be, these technologies can still present challenges to end-user developers, including those with limited or even no experience in electronics and programming.

To provide end-user developers with tools that meet their needs, we need to know what difficulties they face during development and how they naturally attempt to overcome these problems. The work described in this thesis, which was informed by user research undertaken in other end-user programming/development domains and followed a user-centred approach, addresses this gap, and enables us to identify how we can support end-user developers in overcoming their problems and achieving their physical computing ambitions.

1.1.7 A physical card-based tool to support end-user developers' troubleshooting

A key challenge in supporting physical computing development lies—like the challenge for end-user developers *learning* in this domain—in the fact that it involves both *hardware* (electronic circuit construction) and *software* (programming). As I will demonstrate in the coming chapters, end-user developers experience problems in both of these aspects of development, but also in interfacing between them.

There are a number of ways in which end-user developers might be supported in developing and troubleshooting physical computing prototypes. As *active users* (Carroll and Rosson 1987), it has been shown that end-user programmers benefit most from support situated within their tasks (Carroll 1998), and several software tools have been developed to support or scaffold their programming and/or debugging, in applications such as spreadsheets and web mashup environments (e.g., Cao et al. 2015). More recently there has been research to develop hardware tools (e.g., Drew et al. 2016; Wu, Shen, et al. 2017) for visualising otherwise hidden aspects of electronic circuits, while solutions such as basic troubleshooting checklists aimed at makers operate outside of technology-based platforms (Craft 2013).

The medium that I chose for developing a tool to support/scaffold end-user developers' troubleshooting of bugs during development—a physical deck of cards—was inspired by popular creativity and design support tools such as IDEO's popular Method Cards ('IDEO Method Cards' n.d.) and the Tiles IoT ideation toolkit (Mora, Gianni, and Divitini 2017). Physical cards have been used in a number of domains, for example, to support game design (Wetzel, Rodden, and Benford 2016; Mueller et al. 2014), consideration and discussion of information privacy / data protection issues (Luger et al. 2015), and the design of children's technology (Bekker and Antle 2011). However, the tool I describe in this thesis is, to my knowledge, the first application of physical cards for supporting end-user developers in troubleshooting physical computing problems. Besides being a flexible, low-tech medium familiar to most people, physical cards provide other potential benefits to end-user developers, as a diverse population with diverse needs, working styles and goals. This thesis describes how such a tool might be designed and developed, informed by both initial empirical work and the literature, and reports an evaluation of its use by end-user developers, providing insights into the opportunities and challenges it affords.

1.2 Research questions

The overarching research question that guided the work in this thesis was:

How can end-user developers be supported in overcoming problems they experience when developing physical computing artefacts?

The aim of the work I will describe extends upon research in other end-user development domains, to increase our knowledge of this particular population, and determine how they can be supported in their physical computing development activities, specifically in overcoming the most significant problems which arise during development.

In the process of meeting this aim, four main research questions² were developed:

TRQ1: What problems do end-user developers experience when developing a physical computing artefact? (Chapter 3)

TRQ2: How do end-user developers troubleshoot the most significant problems that arise during development, and from what support might they benefit? (Chapter 4)

TRQ3: How can we design a deck of physical cards to support end-user developers in troubleshooting physical computing problems, particularly circuit bugs? (Chapter 5)

TRQ4: What role might a card-based tool play in supporting end-user developers in the process of troubleshooting circuit bugs in a physical computing prototype? (Chapter 6)

Over the course of the coming chapters, each of these thesis-level research questions will be broken down further into sub-questions and addressed.

² These thesis-level research question numbers are prefixed with ‘T’ to differentiate them from research questions defined in respect to individual studies

1.3 Summary of contributions

This thesis provides four main contributions, summarised as follows. Each is described in greater detail in the final chapter.

Contribution 1:

Empirically grounded knowledge of the problems experienced by end-user developers when constructing and programming a physical computing device, including the types, location and number of problems, and whether they are overcome. (TRQ1)

By addressing TRQ1, I gained knowledge of what difficulties end-user developers experience when developing physical computing artefacts. I identified the types of problems that arise, and the specific aspects of physical computing that prove particularly difficult and which have the most significant impact on development success, thereby establishing *where* support might be best targeted, i.e., the troubleshooting of circuit bugs.

Contribution 2:

Empirically grounded knowledge of the behaviours of end-user developers when troubleshooting physical computing problems, particularly circuit bugs, as well as suggestion for support from which they might benefit. (TRQ2)

By addressing TRQ2 I gained knowledge of end-user developers' natural behaviours when troubleshooting circuit bugs, for example, what tactics they employ and whether these are effective. I discovered that end-user developers, like end-user programmers in other development domains, often use unproductive troubleshooting tactics, and would benefit from specific types of support, and in general, being more thoughtful/reflective when troubleshooting.

Contribution 3:

A novel, physical card-based tool to support novice end-user developers in troubleshooting physical computing problems, particularly circuit bugs. (TRQ3)

The knowledge and insights gained from answering TRQ1 and TRQ2 were used to identify recommendations for supporting end-user developers in troubleshooting their bugs, and to design and evaluate support for end-user developers' troubleshooting of physical computing problems, particularly those relating to circuit bugs (TRQ3). This support was in the form of a

novel, physical card-based tool, inspired by creativity support card decks. The tool aims to provide end-user developers with ideas for troubleshooting tactics and encourage them to think/reflect more during troubleshooting.

Contribution 4:

Insights into how a card-based support tool might be used and perceived by end-user developers when troubleshooting circuit bugs in a physical computing prototype. (TRQ4)

An evaluation of this novel card deck delivers insights into the role that such a tool might play in the troubleshooting process, and how it might be perceived by novice end-user developers.

1.4 Research Scope

This research in this thesis investigates how adult end-user developers might be supported in overcoming problems which arise during the development of a physical computing prototype.

Excluded from this work are considerations of problems that other groups or populations may experience in this domain, for example, children or young people, or professional developers of physical computing devices and systems.

While the first study in this thesis (Study 1A)—an exploratory investigation of problems that end-user developers encounter—looks at *all* problems experienced by the participants during a given task, the second study (Study 1B) focuses primarily on the troubleshooting of *circuit bug*-related problems. An in-depth analysis of how end-user developers troubleshoot *program bugs* in physical computing devices is out of scope for this work.

While end-user developers might also benefit from support in other aspects of physical computing development, for example, the ideation of solutions, or in how to go about developing a physical computing device, this thesis focuses only on *troubleshooting* support. See section 7.3 for further discussion of the decision to focus on troubleshooting.

Finally, while there are many potential ways in which support could be provided to end-user developers, this thesis does not claim to have established that the approach I took—a physical card-based support tool—is the most effective. Establishing the best medium for supporting

end-user developers in physical computing development is again outside of the scope of this thesis. Taking into consideration the insights gained through the initial studies, this is just one way in which this type of scaffolding might be presented.

1.5 Methodology and approach

My PhD research and prior to it, the research I conducted for my MSc in Human-Centred Systems dissertation project (Booth and Stumpf 2013), were both inspired by in-the-wild encounters with end-user developers in a Women’s Technology and Arts group, initially as a participant in introductory physical computing workshops and later as an assistant instructor. Through these workshops, and in casual conversations with attendees who were less confident in their programming and electronics abilities, I became aware that the promised easy entry into physical computing, via platforms such as Arduino, might not be easy for *everyone*, particularly those with little or no training or experience in programming, electronics, or both.

1.5.1 An empirical, user-centred approach

Throughout my career I have been a staunch advocate for a user-centred approach (Sharp, Preece, and Rogers 2019, 47–49) to the design and development of user-facing technology, therefore the adoption of a user-centred methodology for investigating this real-world problem, with a view to developing support for end-user developers in their physical computing ambitions, seemed only natural—focusing my research from the outset on actual users (end-user developers) and their tasks meant that any subsequent design work would be well-grounded in an understanding of real-world needs and behaviours, rather than based on speculation and assumption about where difficulties may lie.

I therefore began with rigorous empirical user research to understand the landscape of end-user developers’ physical computing development problems and identify requirements for supporting end-user developers in this domain. Thereafter, I applied my findings to the design and development of a support tool prototype—again involving end-user developers in the process. I subsequently evaluated this tool in a further user study with end-user developers.

When encountering User-Centred Design as an approach within Human-Computer Interaction, it is more often than not in respect to the development of technology-based artefacts—software, hardware, or both. The tool I have created, described in Chapter 5, is designed to support end-user developers when they are *using technology*—both software (programming) and hardware (microcontroller-based circuit construction)—*to create technological artefacts*, but does not do so through the medium of technology.

When I set out on my PhD journey, I anticipated that the destination would be likely to involve designing *something* to support end-user developers in physical computing development, or to make physical computing development easier for them in some way, but had no solid preconception of what form this might take. Through my initial literature review, I became aware of several software-based tools developed to support end-user programming—including debugging—but found no tools to support end-user developers in the construction or troubleshooting of electronic circuits, or in physical computing development more generally. During the course of my research, I became aware of new work to support learners in troubleshooting (debugging) electronic circuits, however, both programming support tools and electronics support tools have relied heavily on some kind of automated analysis of what was being developed. Observing end-user developers, first-hand, struggling to develop and troubleshoot physical computing prototypes, experiencing problems with both hardware and software, led me to consider whether support provided via an alternative medium might present some advantage. Therefore, the choice of a physical card-based medium for the support tool flowed organically from the findings of my initial studies—a decision driven by a user-centred consideration of the needs of end-user developers, particularly novices, informed by an in-depth understanding of the challenges they face, gained through empirical work.

As is customary within a User-Centred Design process, my research followed a staged approach, with each stage informing the next, and with representative users involved at each stage.:

- 1) An **exploratory study** (Study 1A, Chapter 1)
- 2) Deeper analysis of a specific subset of the same data, resulting in a **set of requirements** (Study 1B, Chapter 4)
- 3) Design activities to **develop a prototype** instantiating these requirements (Chapter 5)
- 4) **Evaluation of the prototype** in a final user study (Chapter 6).

1.5.2 Research methods and methodological stance

As a researcher, I am most interested in creating knowledge that can be *used*. To determine how to support end-user developers in physical computing development I formulated research questions that I felt would lead to *useful*—actionable—insights into the difficulties they experience, and their support needs.

Answering these questions meant collecting a range of data—both qualitative and quantitative—and employing a variety of research methods. Broadly, the empirical studies described in this thesis were designed and conducted using a mixed methods approach to data collection and analysis, reflecting a *Pragmatist* worldview—focusing on “*what works* rather than what might be considered absolutely *true* or *real*” (Frey 2018), and using whatever methods would provide “the best understanding of a research problem” (Creswell and Creswell 2018, 48). Methods do, however, still carry with them assumptions regarding what is or isn’t of value to this process (Mertens 2017, 20), and as a researcher, I am conscious that my views about ‘what works’ have been shaped by my own knowledge and experiences, not only through academic study but also professional education and practice, as I discuss further in section 7.4. The methods employed at each stage were as follows.

Understanding the problem and generating requirements: Studies 1A & 1B

Study 1A (Chapter 3) sought to gain insights into end-user developers’ *difficulties* in physical computing development. A commonly used HCI research method for identifying specific usability problems—difficulties—in the use of technology is *user-based testing*—observing “representative users attempting representative tasks” (Lazar, Feng, and Hochheiser 2017). As this was, to my knowledge, the first in-depth investigation of end-user developers constructing and programming physical computing prototypes, I conducted an exploratory study—a first-hand observation of end-user developers undertaking a given physical computing development task while thinking aloud. Through analysing a specific subset of the same data, Study 1B (Chapter 4) aimed to identify participants’ troubleshooting behaviours, determine the effectiveness of these, and from these the findings establish some requirements and/or recommendations for support. Unlike more typical troubleshooting/debugging studies—including the one described in Chapter 6—these participants were dealing with failure resulting from bugs that they themselves had organically introduced during development.

A mixture of qualitative and quantitative data was collected and analysed, including task videos, program files, digital images, and questionnaire data. In these two studies, rather than following a specific, major mixed methods study design (such as one of those suggested by Creswell and Plano Clark (2011, 73), although some characteristics are shared with *convergent parallel* design within that typology), qualitative and quantitative approaches were blended in a multi-layered series of transitions, moving back and forth between analysis modes and within the data, in an “iterative, cyclical approach to research”—a core characteristic of mixed methods research (Teddlie and Tashakkori 2010).

- Quantitative data were captured through questionnaires measuring aspects of participants’ backgrounds, including self-efficacy and self-ratings of their expertise.
- Qualitative data, such as written transcripts of the task videos (verbal protocol of the think aloud, and descriptions of participants’ actions)—in conjunction with the videos themselves—were categorised using inductive and deductive coding methods.
- The categories (codes) were also then used to *quantify* these data (Srnrka and Koeszegi 2007)—transforming it into numerical data that could be used in quantitative analysis, for example, to summarise (e.g., frequency of problem types instances), compare (e.g., frequency of tactics used by successful/unsuccessful participants), or to look for significant relationships (e.g., correlation between problems experienced and self-ratings of expertise).
- Quantitative findings were also used to highlight or identify parts of the data for further qualitative analysis—for example, the discovery in Study 1A that circuit bugs were by far the most frequent cause of task failure, led directly to Study 1B’s focus on analysing the troubleshooting behaviours of participants dealing with circuit bugs. This aligns with what has been referred to as an integrated *elaboration* model (Srnrka and Koeszegi 2007; Mayring 2001), allowing “the problem under investigation [to] be more exhaustively elaborated”.

Rather than adhering to a specific qualitative analysis method in full, analysis of the qualitative data followed a general qualitative approach that broadly aligned with the integrated *generalization* design outlined by Srnrka and Koeszegi (2007), involving stages common to a number of qualitative methods (material sourcing; transcription; unitization; categorization and coding), and with systematic rigour in application, including reliability and validity checks

throughout. Analysis focused on *categorising* data—with a view to *describing* it, either by *deductively* recognising pre-existing concepts drawn from—or inspired by—the literature, or developing new categories or schemes *inductively* from the data, by identifying and naming patterns of thinking and/or behaviour. The process began with immersion in the data, and coding was iterative, involving frequent revisiting and comparing of previously coded data. In these respects, the approach therefore shares some characteristics with methods such as *Content Analysis* (Krippendorff 2012) and *Thematic Analysis* (Braun and Clarke 2006); like Content Analysis, it also involved unitisation and quantification of categorised data. However, the intention was not to develop complex, multifaceted themes, or seek deep meaning through interpretation, and while in some qualitative methods, analysis focuses only on a textual *verbal* protocol, or on communication in some form, in these studies, participants' *actions* were also analysed, for example, their prototype development activities. Again, analysis was driven by what was felt to *make sense* in terms of the research questions, would be *useful* to the overall aim of the research and was *practical* within the constraints of the data I was able to capture.

Support tool development

Designing and developing the deck of troubleshooting support cards (described in Chapter 5) also followed an iterative process, as is typical of a user-centred approach to design (Sharp, Preece, and Rogers 2019, 49). Knowledge gained through studies 1A and 1B resulted in identification of the most significant problems affecting end-user developers' development success, as well as a set of suggestions—loose requirements—to guide the development of the support tool. These findings were then used, along with a targeted review of academic and non-academic literature on troubleshooting, debugging and problem solving, to compile a list of potential troubleshooting tactics for inclusion in the support tool. A similarly targeted design review of card-based tools was conducted, from the academic literature and other sources, including commercial card decks, and from this, a number of key considerations for the design of a card-based tool identified. These, along with findings from the empirical studies, were then used to create an initial prototype of a support tool—a set of troubleshooting tactics instantiating support in the form of physical cards. A small, proof-of-concept study, involving two end-user developers using these cards in troubleshooting tasks, established the need for further design work, and that it would be better to focus on novice end-user developers as the target user group for the support tool.

In two focus groups with novice end-user developers, typical UX methods then were used to elicit feedback from representative users on several design variants and some options for content and form, as well as to test a potential information architecture for the deck. Findings from these focus groups were used to further refine the design of the cards and their categorisation.

Evaluating the support tool

The final stage of the work in this thesis involved trialling the support tool prototype in a user study with twenty novice end-user developers. A within-subjects study design was used (Lazar, Feng, and Hochheiser 2017, 49)—participants undertook two troubleshooting tasks, one with, and one without the tool—enabling comparison of participants’ performance and experience with and without the support tool. Evaluation was therefore based on first hand use/experience of the tool by representative users in a representative task.

This piece of user research also used a mixed methods approach, collecting and analysing both qualitative and quantitative data, although, analysis followed a different approach than in the previous studies.

Task performance was measured, providing quantitative data about troubleshooting success with and without the support tool. However, in a pivot from the previous studies’ predominant focus on analysing qualitative data captured during task execution, the main focus of analysis in *this* study was upon participants’ *subjective feedback* about the support tool, captured via a questionnaire and a semi-structured debriefing interview, following its use. Rich data, in the form of textual transcripts of the video-recorded interviews were subjected to a thematic analysis (Braun and Clarke 2006), using primarily inductive coding to explore these data, and elicit themes in relation to participants’ *use* and *experience* of the support tool. Themes were shaped and refined through immersion in and repeated iteration over the interview dataset. Not only can the findings be used to inform a further design iteration of this particular support tool, but they also—most importantly, in terms of the aim of this study—provide rich insights into the potential for such a tool to support novice end-user developers in this domain.

1.6 Thesis outline

This thesis is structured as follows:

Chapter 1: Introduction (this chapter)

This chapter introduces the research described in this thesis, including the domain and population of study, first providing a motivation for the work. It states the research questions addressed and outlines the academic contributions of the work.

Chapter 2: Related work

An overview and discussion of research undertaken by others, to-date, in related areas of the literature, including the problems of non-experts with programming and circuits, and the work which has addressed supporting them.

Chapter 3: Problems experienced by end-user developers in a physical computing task (Study 1A)

This describes the first research study conducted for this thesis, investigating the problems faced by end-user developers when developing a physical computing device—an observation of 20 adult end-user developers of varying expertise, developing an Arduino prototype to a given specification but without any further instruction or guidance.

Chapter 4: How end-user developers troubleshoot circuit bugs (Study 1B)

Further analysis of data collected during the first study, this time diving deep into the troubleshooting behaviours of participants dealing with failure due to circuit bugs which they had introduced when building the physical computing device. Findings lead to suggestions for support from which end-user developers might benefit when troubleshooting in this domain.

Chapter 5: Developing a physical card-based tool to support end-user developers' troubleshooting

This chapter describes the design and development of a novel, card-based tool to support end-user developers in troubleshooting physical computing development problems. I present a rationale for providing support in this medium, and some considerations for designing cards, informed by a review of the literature, followed

by description of the design process, the tool itself, and the card production process.

Chapter 6: Evaluating the troubleshooting support cards with novice end-user developers (Study 2)

I describe and report findings from an evaluation of the card-based support tool, in a within-subjects user study involving twenty novice Arduino users—adult end-user developers—who each undertook troubleshooting tasks with and without the tool.

Chapter 7: Discussion, limitations, and future work

A final discussion of, and reflections upon, the work described in this thesis. The research questions are revisited, and the contributions are described in full, in relation to the literature. The main limitations of this work are described, the approach and methods used are reflected upon, as is the decision to focus on troubleshooting, and finally, some areas for future work are suggested.

Chapter 2

Related Work

Physical computing involves both programming and electronics. Until the work in this thesis, little was known about end-user developers in the physical computing domain, however, there have been decades of research investigating the problems and typical behaviours of both novice and end-user programmers, and finding ways to support them in creating, modifying, and debugging software.

In this chapter, the literature in areas related to my work, is reviewed and discussed, in the following order:

- 2.1 **End-user developers**—characteristics of end-user developers in comparison to novice, expert and end-user programmers, and the disadvantages they face.
- 2.2 **Non-experts' problems in programming**—common difficulties, the causes of software error, learning barriers, and problems with strategies.
- 2.3 **Non-experts' problems with circuits**—including difficulties observed in learning circuit theory.
- 2.4 **Problems affecting end-user developers in physical computing**—including recent evidence from e-textiles/STEM education, echoing my own findings.
- 2.5 **Supporting end-user developers**—challenges in supporting this population, and the case for situated support.
- 2.6 **Supporting non-expert programmers**—approaches that have been taken to make programming easier.
- 2.7 **Supporting physical computing development**—approaches that have attempted to make programming and circuit construction easier, including modular platforms.
- 2.8 **Supporting troubleshooting and debugging**—troubleshooting as a process; work to support end-user programmers in debugging; the current lack of support for end-user developers' debugging in physical computing, and recent work to address this.

2.1 End-user developers

The term ‘end-user programming’, originally popularised by Nardi (1993), usually refers to the writing of software programs by end users, rather than professional programmers. Nardi made the distinction that for this type of programmer, the act of programming is not the end in itself, but the means by which they achieve the things they want to do, in their work and hobbies. End-user programming has since been studied extensively in a number of different domains, for example, spreadsheets (e.g. Kissinger et al. 2006), web development (e.g. Kuttal, Sarma, and Rothermel 2013) and intelligent user interfaces (e.g. Kulesza et al. 2009). This thesis focuses on the development (and troubleshooting) of physical computing artefacts by end-user developers, which encompasses both programming *and* circuit construction. To recall, the definition I use in this thesis, adapted from Lieberman and colleagues, is as follows:

“End-User Development, in a physical computing context, is a set of methods, techniques, and tools that allow users of hardware and software systems, who are acting as nonprofessional developers, at some point to create, modify, or extend a physical computing artifact.”

(adapted from Lieberman et al. 2006 p.2)

I now turn to the literature that positions end-user developers within the broader field of development.

2.1.1 End-user vs novice and professional programmers

Whereas professional programmers are paid to create and maintain software for others to use, end-user programmers write programs to support their own goals, “in their own domains of expertise”, to use themselves. *“End user programmers might be secretaries, accountants, children, teachers, interaction designers, scientists or anyone else who finds themselves writing programs to support their work or hobbies.”* (Ko et al. 2011).

It is important, say Ko and colleagues, not to conflate end-user programming (or end-user development) with a lack of programming expertise—not all end-user programmers are novices (although some certainly are). A professional programmer may develop software for their own personal use, and in this be considered an end-user programmer; equally, someone not employed as a professional programmer may have previous programming experience that they

draw upon to write programs to support some aspect of their primary work, or personal interests. Intent, rather than expertise, they state, should therefore be the defining characteristic.

Defining end-user development in terms of expertise or area of application, argue Burnett and Myers (2014), risks “siloing” end-user developers and their tools from the learnings from, benefits of, and advancements in, professional software development. Defining it, instead, as a *role*, based on a dimension of *intent*, means that research can consider and address the needs and practices of end-user developers at different levels of expertise, and in different areas of application. Accordingly, the end-user software engineering (EUSE) research area seeks to bring considerations of *quality* into end-user development, for example, by encouraging end-user developers to engage in professional software development practices such as testing (Burnett, Cook, and Rothermel 2004; Burnett 2009; Burnett and Myers 2014; Ko et al. 2011). The work I describe in Chapter 3 and Chapter 4 provides further evidence in support of this position.

Pragmatically, when studying end-user developers, expertise is still a crucial factor to consider, as I discuss further in the next section. Some end-user developers *are*, of course, more experienced than others, and we can probably agree that someone *without* professional experience of developing physical computing devices may have greater support needs than someone who does. Therefore, the scope of the definition of end-user developer I use within this thesis—particularly when recruiting for the studies I undertook—*does not* include individuals currently (or previously) involved in *professional* physical computing development, even if they do, at times, develop physical computing devices for their own personal use. And while my initial studies involved end-user developers of varying levels of expertise, the novel support tool I developed, which I describe in Chapter 5, was designed with *novice* end-user developers in mind, by which I mean *less-experienced* end-user developers.

It is worth, however, noting the distinction between the terms *end-user programmer* and *novice programmer* within the literature. Academic literature on programming, particularly since Nardi’s seminal book on end-user programming (Nardi 1993), typically uses the term *novice programmer* to refer to those who are learning to program with the aim of becoming *professional* or *expert* programmers—that is, those learning with the intention of developing or improving their mastery of programming as a craft. Whereas, according to Nardi and others, an *end-user programmer* is programming to achieve some other end and, if inexperienced in programming, may have little interest in learning the craft of programming, but instead be more concerned with the final result (Nardi 1993; Ko et al. 2011). While this may not always be true in

respect to end-user development in physical computing (see section 2.1.3), a focus more on the *outcome* or *output* of development has important implications for both learning and support, as I discuss further in section 2.5.1 (The challenge of supporting end-user developers).

2.1.2 The disadvantages of non-experts

It stands to reason that when it comes to developing software, non-experts have it harder than their more expert counterparts. Certainly, in many aspects of programming and software development, expert or professional programmers have significant advantage over inexperienced or less-experienced programmers, regardless of whether the literature would class them as novice programmers or end-user programmers. Experts have considerably more programming (and software engineering) knowledge and experience to draw upon, not only in planning, creating and debugging their own programs, but also, for example, when adapting or borrowing opportunistically from programs created by others—a common practice for end-user programmers/developers (Brandt et al. 2009; Rosson and Carroll 1993; Lau et al. 2021). Pertinent to the consideration of *quality* mentioned in the previous section, experts and professional programmers also have a better understanding of how to test the reliability and accuracy of their program results. This knowledge and these skills can take years, and much effort, to acquire. Programming is hard, even for experts, and can be cognitively challenging to learn and master. My work provides evidence that end-user development is challenging in physical computing too, but also that expertise in one aspect of physical computing development, for example, professional programming or engineering experience, does not always translate into fewer problems, more effective troubleshooting, or greater success for end-user developers in this domain.

2.1.3 End-user developers as *makers*

End-user developers of physical computing artefacts—who I also refer to as *makers*—are a heterogeneous population, coming from all types of backgrounds and occupations, for example, artists, designers, hobbyists and researchers, and as a result, their skills and needs are varied. As with end-user programmers, not all end-user developers in this space are novices. A maker may be a professional programmer, but still be inexperienced in electronics, or they may come from an engineering background but have little programming experience. Others may have considerable knowledge and skill in both programming and engineering, but not be

employed to develop physical computing devices. However, some end-user developers lack experience in *both* programming and electronics, and they, of course, potentially face some of the biggest challenges, particularly if they are more interested in the *output* of development—the resulting artefact—than in acquiring the knowledge and developing the skills needed to produce it.

However, unlike in some end-user programming domains, *making* is not necessarily always just getting about the job done—in this domain, an end-user developer may have motivations beyond realisation of the end product. Hands-on making can be a pleasurable activity in itself, and the act of physical creation satisfying (Tanenbaum et al. 2013), while much is made also of the social and community aspects of these practices (Kuznetsov and Paulos 2010). Intrinsic motivation has a positive effect on performance in learning to program (Bergin and Reilly 2005), and *bricoleurism*—tinkering, making and fixing things—has been shown to correlate with intrinsic motivation in end-user programming (Aghaee et al. 2015). If end-user developers are *intrinsically* motivated to create physical computing devices, this may provide a boost in their efforts to learn the skills to do it, stick at it, or to see any problems they encounter through to resolution.

The next section will discuss some of the work from the considerable body of research investigating the problems of novice and end-user programmers.

2.2 Non-experts’ problems in programming

Much is known about the problems of inexperienced programmers. Novices’ numerous difficulties in learning to program—and in the various activities involved in programming—are covered extensively in the Computer Science literature, spanning several decades of work. While discussing all of these problems, or the work of all those who have investigated them, is beyond the scope of this thesis, a small sample of the literature provides some flavour of the difficulties that others have observed in this field.

There is consistent evidence that novice programmers can struggle with concepts and activities that are *fundamental* to programming. For example, Spohrer and Soloway (1986), studying the mistakes of novice programmers, note problems with learning and understanding the semantics

of constructs—even the most basic ones, for example, variables—and applying them appropriately. They also report novices’ difficulties in interpreting problems and tasks, and in composing plans—as well as a tendency to rely upon existing knowledge when doing so. In a longitudinal study, Garner, Haden, and Robins (2005) asked teachers of introductory Computer Science to report problems that students sought assistance with—the main difficulties were with the basic mechanics of programming (e.g., basic syntax), program design, basic program structure, and problems understanding what a program is supposed to do; students also experienced conceptual and implementation problems with arrays, arguments/parameters and return types/values, loops, constructors, and control flow. Problems with variables, arrays and loops also feature in the common novice programmer mistakes described by du Boulay (1986).

A similar set of problems was reported in a study by Lahtinen, Ala-Mutka and Järvinen (2005), asking 559 students and thirty-four teachers across six universities what they had found and observed, respectively, to be difficult in learning to program. Notably, these authors also found that students considered *debugging* to be the most difficult activity in learning to program and *error-handling* the most difficult *concept* to learn. The ability to debug faulty programs—to locate and resolve the causes of error (Katz and Anderson 1987)—is crucial to programming success, and there is much work attesting to the difficulty of this type of problem solving, not only for novice and end-user programmers but even experts, and investigating ways to support them (see sections 2.2.3, 2.8 and 2.8.1 for further discussion).

While it would be reasonable to assume that novice end-user developers will face similar—or analogous—difficulties when learning to develop and troubleshoot physical computing prototypes, the work in this thesis provides the first evidence of this, in respect to both programming *and* circuit construction.

2.2.1 Causes of software error

Perkins and Martin (1986) suggest that novices’ problems with programming stem from *fragile knowledge*—including knowledge that is *missing* (not yet learned), *inert* (known but failed to be retrieved), or *misapplied* in some way—in combination with *poor problem-solving strategies*. Subsequent work by Ko and Myers (2005) focusing on the causes of software errors, based on research into human error (Reason 1990), has suggested that errors are due to *cognitive breakdowns*, in which programmers (including end-user programmers) encounter problems in the application of skills, rules, or knowledge. Ko and Myers suggest that breakdowns can be

investigated by classifying the *action* being performed, the *interface* the action is performed on, and the *information* being acted on—in the first study in this thesis (Chapter 3), analysis of end-user developers’ problems was informed by this approach, looking at breakdowns, and the locations in which these occurred.

2.2.2 Learning barriers

One way to classify problems that can lead to error, seen frequently in the end-user programming literature, is in terms of *learning barriers* (Ko, Myers, and Aung 2004). In a study investigating problems experienced by end-user programmers learning Visual Basic, the authors identified six frequently encountered barriers—obstacles which a learner must overcome in order to make progress. When encountering these barriers, learners’ invalid assumptions—caused by knowledge breakdowns—can result in error, stalled progress, or further barriers.

- **Design** barriers—“*I don’t know what I want the computer to do...*”
- **Selection** barriers—“*I think I know what I want the computer to do, but I don’t know what to use...*”
- **Coordination** barriers—“*I think I know what things to use, but I don’t know how to make them work together...*”
- **Use** barriers—“*I think I know what to use, but I don’t know how to use it...*”
- **Understanding** barriers—“*I thought I knew how to use this, but it didn’t do what I expected...*”
- **Information** barriers—“*I think I know why it didn’t do what I expected, but I don’t know how to check...*”

Most of the barriers these authors observed were *Understanding* barriers—participants struggled to diagnose unanticipated behaviour or failure at compile-time or runtime, often a result of error. *Use* barriers were also prevalent, for example, participants’ difficulties in determining correct syntax, as were *Coordination* barriers, in which working out the ‘invisible rules’ of combining programming interfaces proved challenging.

Relationships between barriers were also observed, with invalid assumptions made when overcoming barriers leading to new barriers—overcoming *Coordination* barriers frequently resulted in further *Understanding* or *Use* barriers; *Design* barriers led frequently to *Selection* barriers or *Coordination* barriers; *Selection* barriers led to *Use* barriers and *Use* barriers tended to result in *Understanding* barriers, which in turn led to *Information* barriers. These “common paths of failure”—chains of breakdowns—and their impact upon performance, point to value in

identifying where such obstacles may lie in systems and environments used by end-user developers, and investigating whether there are opportunities to mitigate or reduce these through targeted support and/or better design of development tools. In the first study of this thesis (Chapter 3), analysis of problem locations provides insight into where support for end-user developers might best be targeted.

Learning barriers have been identified in other end-user programming domains—albeit differing in frequency, which suggests that factors such as the type of environment or task may affect which obstacles (or chains of obstacles) end-user programmers will encounter in a particular programming context. For example, *Selection* and *Coordination* barriers proved most prevalent in studies of end-user programmers debugging intelligent agents (Kulesza et al. 2009) and machine learning in spreadsheets (Sarkar et al. 2015), while in a study of end-user programmers programming web mashups with Yahoo Pipes, *Understanding* and *Use* barriers were most common (Kuttal, Sarma, and Rothermel 2013). Although they do not report the frequency of barriers, Cao and colleagues, in an end-user web mashup programming study using Popfly, found a close tie between *Design* barriers and unproductive framing episodes—where users try to understand a problem, but fail to generate an idea for action—suggesting that these barriers frequently lead to an inability to make progress in this environment (Cao et al. 2010).

2.2.2.1 Learning barriers in programming environments for physical computing

There is already some evidence of learning barriers, for end-user programmers, in *programming environments* for physical computing (Booth and Stumpf 2013). In this study, conducted for my MSc dissertation project, I compared the benefits and effects of textual and visual programming environments for Arduino, for end-user developers new to Arduino. I found *Use* barriers to be the most frequently observed barrier type in both of these environments, highlighting that end-user developers programming Arduino devices can struggle to use the basic building blocks of the language correctly. Additionally, *Understanding* barriers were common in the textual environment—participants struggled to understand system feedback within the environment—and *Selection* barriers were common in the visual environment—participants had difficulty deciding which of the available programming constructs would achieve what they wanted to do. Again, these results suggest that the type of environment can affect the type of obstacles experienced.

While the study provides some insight into barriers faced by novice end-user developers when *programming* a physical computing device, the electronic circuits used in the tasks were prebuilt, therefore, there was no analysis of problems relating to circuit construction—participants only *programmed* the circuits; they did not create or modify them. In the first study of this thesis, described in Chapter 3, I focus on identifying the problems—including obstacles (barriers)—that end-user developers experience in a development task involving both programming *and* circuit construction.

2.2.3 Strategies

Developers adopt different *strategies* for solving programming problems, for example when debugging. Unsurprisingly, novices and experts can differ in their use of debugging strategies (Nanja and Cook 1987; Vessey 1985), and novice programmers' poor choice and application of strategies can lead to unproductive troubleshooting behaviours (Murphy et al. 2008). Strategies adopted by less-experienced programmers may even be destructive, for example, Perkins and colleagues found that students who choose to tinker unsystematically when debugging—making lot of changes in the hope of fixing bugs—often introduce new bugs, making programs worse, rather than improving them (Perkins et al. 1986). Investigating the strategies that developers use can provide insight into the *challenges* they face in applying them, and help identify opportunities to address these difficulties (LaToza and Myers 2010). Equally, comparing successful and unsuccessful debuggers can help to identify the types of approaches—thinking—that might be encouraged in order to improve debugging skill (McCauley et al. 2008).

Following from this, knowing what strategies—or patterns of behaviour—end-user developers naturally employ when problem solving, and whether these are successful or lead to additional problems, can help us to identify what approaches are problematic for end-user developers, what strategies they lack, and whether they might be guided towards particular or more productive behaviours, in order to help them solve their problems more efficiently and successfully (Grigoreanu, Burnett, and Robertson 2009).

To this end, several studies have looked at end-user programmers' debugging strategies. A study by Kissinger and colleagues analysed end-user programmers' information gaps when debugging spreadsheets and found 30% of these to be in respect to what strategy to use (Kissinger et al. 2006), for example, how to find or fix errors, or how to test their spreadsheets—users were frequently unsure whether values and formulas were correct, or how to establish this. Similar

findings have been reported in other studies, where end-user programmers have been observed to have difficulty knowing how to get started, or generating ideas to help them proceed (Cao, Fleming, and Burnett 2011). When debugging, failure can manifest in a different location from the actual fault, and choice of strategy has been shown to matter in the success of finding these faults (Prabhakararao et al. 2003). Also, certain types of bugs may be localised quicker through particular strategies, however, end-user developers may not have the knowledge or skill to choose the strategies most likely to help them.

Gender differences have been observed in end-user programmers' choice of strategies and their success in using these to resolve bugs. In one study, males were more likely to follow dependencies or use tests, and were more successful when they did so, whereas females inspected their code and checked it against the specification more often, which also resulted in greater success (Subrahmaniyan et al. 2008). While gender does not feature in any analysis of performance or behaviour in this thesis, Subrahmaniyan et al.'s findings suggest, as other have argued (Beckwith 2007; W. Jernigan et al. 2015), that support for end-user development should consider and accommodate different users' different information processing styles.

Finally, the development environment itself can also have an effect on the choice and successful use of debugging strategies. In the Popfly web mashup environment, where blocks representing functionality are, in effect, black boxes, participants in one study had trouble *inspecting* their 'code' (a common strategy), being unable to see the inner workings, and had difficulty understanding the links between blocks when trying to follow dependencies (Cao et al. 2010). This highlights the importance of ensuring that the design of environments aimed at end-user developers supports the problem-solving behaviours that they naturally employ, and of helping end-user developers to adopt strategies and tactics that are appropriate for a particular environment or type of development, and can be used effectively in that context.

Inspired and informed by the work discussed in this section, the study described in Chapter 4, extends our knowledge of end-user developers' problem-solving behaviours into the domain of physical computing. Analysis of how end-user developers troubleshoot circuit bugs during development identifies several common patterns of behaviour, while subsequent examination of the efficacy of these approaches—including a comparison of the tactics employed by successful and unsuccessful troubleshooters—provides insights into how end-user developers might be supported when troubleshooting in this domain.

2.3 Non-experts' problems with circuits

While I had no difficulty finding literature about the problems and behaviours of novice and end-user programmers, or evidence of difficulties that students experience with circuit theory in an educational context (as I will shortly describe in section 2.3.1), I found very little about end-user developers constructing electronic circuits. Although hobbyist electronics kits have been around for many years—for example, the *Heathkit* construction kits that were popular with amateur electronics enthusiasts for most of the latter half of the twentieth century (Brueschke and Mack 2019)—I found no literature on problems or behaviours observed in hobbyist or amateur electronics construction, prior to the advent of the Maker Movement. More recently, however, Wakkary and colleagues report having difficulty using tutorials—a common approach for novice makers—to construct electronic devices, in part due to problems with instructional materials of inconsistent quality, or the assumption of a particular level of knowledge (Wakkary et al. 2015). In the same year that the findings from the study described in Chapter 3 were published (Booth et al. 2016), David Mellis, Leah Buechley and colleagues, investigating how to engage amateurs in the fabrication of electronic devices using PCBs (printed circuit boards), report that participants found it challenging to troubleshoot circuits consisting of low-level components, and struggled to generate hypotheses for the causes of failure (Mellis et al. 2016). This further confirms the need for work to identify and understand the difficulties experienced by non-experts—including end-user developers—when working with circuits.

2.3.1 Circuit theory: problematic for learners

Despite the dearth of literature in respect to end-user developers' problems with circuits, there is, however, plenty of work attesting to the difficulties that children and even much older students have in learning and applying the theories of electricity that are fundamental to understanding and constructing electronic circuits. Numerous studies in the educational literature show that learners struggle to fully grasp important, abstract concepts such as current, voltage (potential difference) and resistance, and in the case of some of these concepts, even fail to differentiate between them—current and voltage, for example, are frequently confused.

Research in this space has identified common misconceptions—alternative views—of how electricity works, that affect students' reasoning about even simple circuits (McDermott and Shaffer 1992). Several alternative conceptual models have been identified in children, including,

for example, that current is stored in a battery and consumed over time, or that current is used up by components in the order of the direction of the current (Shipstone 1984; 1988). Many of these misconceptions are still evident in high school students and university students (McDermott and Shaffer 1992), including those studying to be engineers (Periago and Bohigas 2005), electrical engineers (Métoui et al. 1996) or physics teachers. Even university science teachers have been found to hold some of the same, faulty mental models (McDermott and Shaffer 1992). These preconceived ideas prove resistant to change, even after instruction in the scientific theory (Engelhardt and Beichner 2004).

As has been demonstrated repeatedly in the literature, these misconceptions affect students' reasoning about circuits, and thus, their ability to accurately interpret circuits or construct new ones reliably and successfully. Many students have a great deal of trouble reading circuit diagrams—for example, they have difficulty understanding parallel circuits, and recognising these when represented in different configurations within circuit diagrams (schematics), tending to focus on the lines between components rather than on the electrical connections these represent (McDermott and Shaffer 1992). Equally, while resistance is a crucial part of circuit theory, many students struggle with this concept and are therefore unable to accurately predict its effects on the behaviour of a hypothetical circuit (McDermott and Shaffer 1992).

All of this has important implications for end-user developers' physical computing development, particularly for end-user developers who are less experienced in electronics. While many end-user developers may have been taught electrical circuit theory at school, they may have inaccurate mental models of the fundamental concepts, which might affect their ability to apply these successfully—several studies show evidence of students reverting to their previous, non-scientific views after a period (e.g. Mackay and Hobden 2012). As a result, some end-user developers may struggle to plan their circuits, or understand existing ones, for example, if wanting to use and adapt something they have found—reuse is a common behaviour for makers (Oehlberg, Willett, and Mackay 2015). If end-user developers do not properly understand the relationships and dependencies between different variables in a circuit and cannot predict the effect of changes to it, they may have trouble diagnosing and fixing problems they encounter. Whatever beliefs end-user developers hold will inform their decisions and actions when developing and troubleshooting their devices.

2.4 Problems affecting end-user developers in physical computing

We know from the literature that programming is difficult for novice and end-user programmers, and physical computing development involves *both* programming and electronics. It also involves coordinating those two types of activity to achieve a particular goal, therefore in order to achieve this, an understanding is needed of how the two relate and can be coordinated, or what coordination between them is required.

Until the early work in this thesis, however, there was little knowledge of the problems affecting end-user developers in this domain. While the large number of user posts asking for help in online communities (e.g., ‘Arduino Forum’ n.d.; ‘Adafruit Customer Support Forums’ n.d.) suggests that people experience a lot of problems when developing physical computing devices, it would be difficult to ascertain which or how many of these problems were reported by end-user developers.

Little was also known about how end-user developers *troubleshoot* problems which arise when they are developing or modifying physical computing devices, or the difficulties they face when doing so, but, again, as we know that debugging programming problems alone is hard, particularly so for those lacking programming knowledge, skills and experience to draw upon when recognising, diagnosing and resolving problems, it is likely to be so too in this domain. It also stands to reason, as others have suggested (Tetteroo, Soute, and Markopoulos 2013), that troubleshooting physical computing problems may be even more difficult for end-user developers, firstly, because there is more to know in order to fully understand the domain, and secondly, with both circuit and program involved, the problem space is more complicated. Research has shown that as complexity *increases*, troubleshooting performance *decreases*—the more components or relationships to consider, the greater the likelihood of misdiagnosis (Morris and Rouse 1985).

Evidence of this has been reported in another domain popular with makers, which similarly involves interactions and dependencies between hardware and software: 3D printing. Here, suboptimal or failed results are common, but can be tricky to diagnose (Ludwig et al. 2014). For example, there may be a mechanical problem with the printer, an error in the configuration of the printer, or in the software that controls it, or some error in the 3D model itself, while

environmental or physical factors can also have an effect, for example, the temperature of the room. These faults and errors can also co-occur and interact, resulting in a tangled knot of failure that the user must analyse and unpick, in order to resolve their problems and achieve a better end result.

In principle, all of this this can apply in physical computing development too. Unexpected or erroneous output may be the result of problems with the physical construction or configuration of the circuit or its individual components (including the microcontroller board), problems with the system hardware or software on the computer upon which the IDE is running and to which the circuit is connected, problems with the IDE application or configuration thereof, problems in the program(s) that the user has written, or with any external programs or libraries they are referencing. And likewise, environmental factors may contribute to failure or unexpected behaviour, particularly where a circuit involves sensors, for example, heat, light, sound, or any physical properties of a user's interaction with a prototype (e.g., skin temperature or speed of movement). Any of these factors, alone or in combination, can cause unexpected or erroneous output or device behaviour.

2.4.1 Existing evidence of problems in physical computing

While much had been written or said about how platforms such as Arduino—originally created for use by designers (see section 2.7.3)—could make developing physical computing devices easier for non-experts, by the start of my PhD research there appeared to have been little investigation into the *problems* or *challenges* that end-user developers might experience in using them.

It does seem that there is still room for improvement in the design of tools within the physical computing development space. As discussed earlier, in section 2.2.2, a study I undertook during my MSc (Booth and Stumpf 2013) found evidence of *learning barriers* in visual and textual *programming* environments for Arduino, including the official Arduino IDE, however this did not look at circuit construction, only programming. There have also been calls for pedagogical tools more suited to the needs of young learners (Blikstein 2015), from researchers/educators concerned about the usability issues that maker-centric platforms present to children, and the potential for 'black box' toolkits to conceal information vital to learning.

Some studies have looked at the problems experienced when creating or troubleshooting e-textiles projects, but these have mostly been in supported learning environments, or have involved children or young people in groups. For example, Jayathirtha and colleagues analysed student interviews and project reports for evidence of debugging problems and found that problems were experienced in both programming and circuitry, just over a third of programming problems could be classed as more complex, and that collaboration was a key resource when debugging (Jayathirtha, Fields, and Kafai 2018). Further work by some of these authors investigated ways to develop students' e-textiles debugging skills (e.g., Fields, Searle, and Kafai 2016; Kafai et al. 2014), however, again, these are in respect to pedagogical environments, usually with a focus on developing curricula and specific interventions to engender STEM learning through e-textiles development.

Since the first study described in this thesis (Chapter 3; Booth et al. 2016)—an empirical investigation of the problems experienced by end-user developers when developing a physical computing device—there has been some work looking at the problems of novices (students) when using Arduino. Sadler and colleagues, recoding data from a previous study by Jung and colleagues (Jung et al. 2014) in which high school students prototyped the most simple Arduino project (making a single LED blink), followed by open-ended exploration, discovered that failures (obstacles which required human intervention) were dominated by circuit wiring errors (all but one of the twenty participants experienced these), while programming errors (nine participants) and design ideas due to misconceptions and knowledge gaps were also reported (Sadler, Shluzas, and Blikstein 2017)—the types of failure reported echo my own findings. More recently, DesPortes and DiSalvo adapted the coding schemes (*problem types* and *problem locations*) reported in Booth et al. 2016 (Chapter 3)—in a qualitative analysis of the problems of university students using Arduino for the first time (DesPortes and DiSalvo 2019). As in my study, participants experienced more software-related problems than hardware-related ones, and problems were observed in both programming and circuit construction, as well as in interaction between the two, again echoing my findings. These studies, like my work preceding them, provide further evidence that novices experience problems with the most basic concepts and tasks fundamental to physical computing development, for example, LED polarity errors and problems with open (incomplete) circuits. While both of these studies involved students, my own work focuses on *adult* end-user developers, and in the first two studies, end-user developers with a wide range of skills—representative of the wider maker population. As I will

show, all end-user developers experience problems when developing a physical computing device, irrespective of their background or level of expertise.

Until my work, however, there was no empirical evidence of these problems, including how end-user developers attempt to troubleshoot their physical computing bugs, or how successful they are in doing so, making it difficult to know where support might be best targeted. Analysing the types of bugs that end-user developers introduce during physical computing development, how these bugs manifest, and whether they are able to recognise them, accurately diagnose the symptoms of failure in runtime behaviour they observe, and fix them, helps to establish where some of the pain points lie. Knowing what troubleshooting behaviours end-user developers employ, in their attempts to overcome problems, helps us to determine what aspects of troubleshooting they could most benefit from help with.

2.5 Supporting end-user developers

It stands to reason that a lack of knowledge about programming and electrical engineering concepts, and the relationships between the two, may limit novice or less-experienced end-user developers in their prototype development ambitions, lead to frustrating bouts of troubleshooting, or even prevent them from completing their projects. Supporting them in overcoming their problems and developing their knowledge, however, is not a simple case of providing the right education or training.

2.5.1 The challenge of supporting end-user developers

Much of the literature addressing *students'* difficulties with circuits and programming suggests pedagogical methods that encourage the development of good mental models (Mayer 1981; du Boulay 1989; Liégeois et al. 2003; Shaffer and McDermott 1992), however, supporting end-user developers in their tasks requires a different approach. Educators can design courses and learning materials to support the incremental development of students' knowledge and skill in programming and electronics (e.g., Buechley, Eisenberg, and Elumeze 2007; Fields, Searle, and Kafai 2016), but end-user developers are often self-taught, and their learning situated in a task they are trying to accomplish—learning is likely to be more ad hoc and less structured than for those in educational settings.

While there is no shortage of books, online tutorials, forums and examples available to guide end-user developers in physical computing development, they may not be able to locate, assess or use these effectively, as my work in this thesis will demonstrate (see Chapter 3 and Chapter 4). Without sufficient domain knowledge to draw upon it may be difficult for an end-user developer to judge the quality or relevance of a resource, recognise whether or how it might be useful, or determine how it might be applied or adapted for their needs. They may also have limited time available for study and/or be subject to a “production bias” that disinclines them from acquiring any more knowledge than they need to get by—*“the paradox of the active user”* (Carroll and Rosson 1987). Blackwell’s Attention Investment theory (Blackwell 2002) also suggests that end-user developers will weigh up the benefits of learning something new, in terms of the cognitive cost of doing so and the risk that it may not be worth the effort, or result in failure—yet another potential barrier to end-user developers’ learning, with implications for how support might be presented.

The case for situated support

End-user programming and end-user development research has taken a pragmatic view of how to support end-user programmers/developers in the activities of programming. Minimalist theory suggests that users get more value from support situated within their tasks (Carroll 1998), and studies have established the effectiveness of providing in situ scaffolding for problem solving in existing end-user programming environments (e.g. Cao et al. 2015), however, until the work in this thesis there had been no research to investigate this in the context of physical computing development. To provide effective, situated support in this domain we must first establish what support end-user developers might benefit from in these activities, starting with the problems they experience that can impact their progress and success, and the difficulties they encounter when trying to overcome them.

2.6 Supporting non-expert programmers

Numerous usability issues have been identified in the design of programming languages and environments for novice programmers, and recommendations made for how to address these. Concerned by the tendency for the design of new programming systems to be driven by technical considerations, rather than taking into account the considerable learnings from

research into novice programmers' issues, Pane and Myers (1996) compiled a summary report of work in this area, with the aim of making this knowledge more easily available to those designing programming languages and environments. However, not all programming systems aimed at non-expert programmers are concerned with improving skill. A comprehensive taxonomy of novice programming languages and environments collated by Kelleher and Pausch (2005) groups them into two broad categories: 1) systems for *teaching*—created to help novices to learn to program and ultimately transition to general-purpose languages and 2) *enabling* systems—created to empower novices or inexperienced programmers to program more easily and quickly. This latter category includes many systems used by end-user programmers that enable them to achieve their task goals.

2.6.1 Making programming easier

Much work addressing novice and end-user programmers' difficulties focuses on *simplifying* programming languages or environments.

A common approach to simplifying programming is through the use of *visual programming languages* (VPLs), which use a visual representation instead of, or in addition to, more traditional textual representations of program source code. Examples of VPLs aimed at end-user programmers include *Forms/3* (Burnett et al. 2001) in the spreadsheet paradigm, *LabVIEW* ('National Instruments LabVIEW' 2013), which uses a box-and-wire notation for instrument control and industrial automation, and *Max* ('About Max' n.d.), which again uses a box-and-wire notation, for programming music and multimedia. Other VPL tools allow end users to manipulate machine learning datasets (Sarkar et al. 2015) or interrogate, explore and interact with web service data (Chang and Myers 2016), both within the familiar visual interface of a spreadsheet. In the popular *Scratch* VPL (Resnick et al. 2009), developed to teach children how to program using animations, programs are created by snapping together graphical blocks of different colours, and the shapes of these blocks determine what they can connect with, thus avoiding syntax errors common in text-based languages. This building-block metaphor has been adopted by several other VPL environments, for example, MIT's *App Inventor* web application, which helps end-user programmers develop Android and iOS applications ('MIT App Inventor' n.d.; Wolber 2011).

There are other approaches, beyond visual programming. For example, *Programming-by-Demonstration* eases the effort of learning a programming language by allowing users to

demonstrate how they would undertake a task, as a means of program creation—the system records the user’s actions as a program, for later reuse and/or modification (Lieberman 2001; Cypher et al. 2010). *Natural Programming* attempts to bridge the gap between humans’ natural language and behaviours in describing problems and their solutions, and the rather less natural instructions usually required to program computers (Pane and Myers 2006).

Some programming environments combine paradigms. For example, *Flip* (Good and Howland 2017) employs both visual programming *and* natural language, enabling programmers to compose game event scripts using visual blocks, while simultaneously providing a natural language representation of the resulting script, which can aid learners in sense-checking and debugging their programs. The use of multiple representations to support novices in learning to program is also a feature of a visual programming environment for physical computing development that I used in a previous study with novice Arduino users (Booth and Stumpf 2013). I will discuss this in the next section, along with other ways in which the visual programming paradigm has been used to make the programming of physical computing devices easier, both for children and end-user developers.

2.7 Supporting physical computing development

Several tools have been developed to make developing physical computing devices easier. Some focus on one aspect of physical computing, be that programming or circuit construction, while others address both.

2.7.1 Easier programming in physical computing

Some research in this area has focused on lowering the bar for *programming* physical computing devices, by providing visual programming environments. For example, Stanford University’s *d.tools* platform enables designers of interactive devices to lay out and connect software duals of smart, physical components (e.g. *Phidgets* (Greenberg and Fitchett 2001)), plugged into a dedicated hardware interface connected to their computer, in a visual statechart-inspired editing environment (Hartmann et al. 2006). They can then use this visual environment

to define the interaction behaviours of their device, test these behaviours, and iterate rapidly through different design ideas. Motivated by their work with d.tools, the same authors developed a tool which employs Programming-by-Demonstration techniques with physical computing devices, using direct manipulation to make the programming of interaction easier and more efficient (Hartmann et al. 2007; Fourney and Terry 2012).

Several visual programming environments have been developed for Arduino. For example, *Modkit Micro* (Millner and Baafi 2011), which I used in an earlier study (Booth and Stumpf 2013), adopts the same blocks-based metaphor as Scratch, as does *Scratch for Arduino (S4A)* ('S4A: Scratch for Arduino' n.d.), allowing users to create Arduino programs by snapping together blocks representing different Arduino code elements. In Modkit, a user can switch back and forth between the graphical view and a text view of their program, enabling them to see how the blocks program translate into Arduino code, and changes made in one view update the other. S4A goes one step further by sending the states of sensors and actuators back to the IDE, where they can be monitored. Both of these environments present a potentially easier route into programming Arduino-based devices, but I have not yet found evidence of significant adoption of either environment, or any other VPL for Arduino, by end-user developers.

2.7.2 Easier circuits in physical computing

Some work has aimed to make it easier to construct the electronics or hardware aspect of physical computing devices. For example, the *Programmable Bricks* created at MIT Media Labs enabled children to easily create physical computing devices by connecting sensors and motors to a computer embedded in a LEGO brick and program them using the Logo programming language (Resnick et al. 1996). A variation of these bricks, the *MIT Cricket*, was developed for use in science and engineering education, to help children create easily devices that they can use in active, hands-on exploration and investigation of concepts (Resnick, Berg, and Eisenberg 2000).

Some tools have been aimed at end-user developers, rather than children. Parallax's *BASIC Stamp*, released in 1992, was a microcontroller board incorporating sensors and outputs, aimed at hobbyist engineers (Benchoff 2015). These boards dominated the hobbyist electronics market for many years, but connecting components to them required soldering, and they could only be programmed in one language—BASIC. Other tools aimed to facilitate rapid prototyping by non-engineers, for example, the *MetaCricket*—an offshoot of MIT's Cricket aimed at designers (Martin, Mikhak, and Silverman 2000), *Phidgets*—'physical widgets', developed to help designers and

programmers build physical interfaces, with minimal electronics knowledge, and no need for soldering (Greenberg and Fitchett 2001), and *Calder*, which allowed designers to control the behaviour of prototypes constructed from modular components—or even traditional circuits constructed on a custom breadboard component—through Macromedia Director (Lee et al. 2004). Gadgeteer (Villar, Scott, and Hodges 2011), also aimed to make it easier for end-user developers to build interactive physical interfaces through plug-and-play hardware components, however, while it eased the creation of physical devices, programming these required knowledge of fully-fledged, general-purpose programming languages and environments such as Visual Studio, perhaps limiting its adoption by end-user developers less experienced and less confident in programming.

2.7.2.1 Virtual circuits

Taking a different approach, some tools enable end-user developers to plan their circuits *virtually* before creating them with physical components. For example, *Fritzing* (Knörig, Wettach, and Cohen 2009), a popular, open source virtual circuit prototyping tool, allows users to graphically lay out circuits on a virtual breadboard (see Figure 16, on page 69, for an example of a virtual circuit created with this tool). Building upon this, *AutoFritz* extends Fritzing with autocomplete functionality, using datasheet schematics and a database of projects from the Fritzing community, to automatically suggest wiring configurations and further modules to add, for inserted components (Lo et al. 2019).

Some tools aim to automate the translation between physical and virtual circuits. *CircuitSense* (Wu, Wang, et al. 2017) translates physical circuits into virtual ones, automatically detecting the placement of components and wires in the pins of a custom breadboard and predicting component identity. A virtual version of the circuit is then visualised in Fritzing's Breadboard view, where users can perform further editing, for example, revising incorrectly identified component types. *CircuitStack* (Wang et al. 2016) addresses the opposite problem, that is, the translation of virtual circuits (breadboard schematics) into physical ones, by sandwiching printed circuit paper and a breadboard between custom PCBs.

Autodesk's *123D Circuits Electronics Lab* web application ('123D Circuits Electronics Lab' n.d.), now discontinued as a standalone product, took virtual circuits one step further, by combining virtual circuit construction with a code editor and a simulator, so that end-user developers could 'upload' their program to their virtual circuit and simulate runtime behaviour. A version of 123D

Circuits now exists within Autodesk's *Tinkercad* environment, online, along with a number of tutorials ('Learn How to Use Tinkercad' n.d.). Tinkercad's 3D modelling tool has proven very popular with makers and young people—I used it to design the cards stand described in Chapter 5—however it is not clear how widely its virtual circuit tools have been adopted.

Finally, some tools have also proposed 'blending' the digital and the physical. Proxino (Wu et al. 2019) uses physical proxies to bridge between virtual circuits and the physical world, allowing circuit designers to test interaction with a virtual circuit via real I/O components.

In theory, virtual prototype construction and simulation. may help end-user developers to identify potential problems with their planned circuits and work out how to rectify these before physically building them, however, I have yet to see formal, empirical studies confirming this. In practice, this approach may even present different issues.

2.7.3 Making circuits and programming easier

Some physical computing development tools tackle the difficulty of both circuit construction *and* programming. A platform that has proven popular with hobbyist roboticists is the *LEGO Mindstorms* robotic construction platform ('LEGO Mindstorms' n.d.), which evolved out of MIT's work with Programmable Bricks. This combines the LEGO Technics system with a programmable 'Intelligent brick' and modular sensor and actuator components. The brick is programmed via a visual programming environment that previously used the LabVIEW engine but is now based on Scratch.

The tool currently most popular with makers—*Arduino* (Mellis et al. 2007; 'Arduino' n.d.)—was originally developed to teach physical computing to designers, but has since become the de facto physical computing platform for many types of end-user developers constructing interactive devices. An open source hardware and software platform, it evolved out of Hernando Barragán's *Wiring* project (Barragán 2004; 2016), which allowed non-experts to easily connect sensors and actuator components to a microcontroller board, without soldering, and to program these circuits in a simple environment based on the *Processing* IDE ('Processing' n.d.). Using standard headers, additional hardware circuit boards, called shields, can be stacked on top of Arduino boards, to extend their capabilities, for example, to add Wi-Fi or Bluetooth communication.

As the Arduino platform is open source, there have been numerous clones, extensions and adaptations of its hardware designs, for example, an offshoot of Arduino, *LilyPad Arduino*, was developed specifically for e-textiles, by Leah Buechley (2005). The LilyPad microcontroller board and components (e.g., sensors and LEDs) are designed to be sewn onto fabric and connected using conductive thread rather than wires. LilyPad was enthusiastically adopted by crafting communities, and has been used successfully to engage otherwise underrepresented groups in STEM activities, for example the creation of programmable wearable, interactive devices (Buechley and Hill 2010). Studies have also shown the potential for e-textiles toolkits and curricula to help learners understand STEM concepts (Peppler and Glosson 2013; Fields, Searle, and Kafai 2016), including some of those mentioned earlier, which are prone to misconception.

Arduino's *relative* ease of use, compared to previous ways of developing physical computing devices, is one of its biggest draws for end-user developers, however, using the Arduino platform still requires some measure of knowledge and skill in programming and electronics. The Arduino IDE—a no-frills, notepad-style editor—has a simple, uncluttered interface that appears to simplify the programming of Arduino-based devices. However, while many refer to the “Arduino language”, the user is, in fact, programming in C/C++ (Williams 2015), albeit with the benefit of additional libraries, referenced as standard within the IDE. These make communication between the computer and Arduino board easier (including the uploading of programs to the microcontroller—previously not an easy task for non-experts), and abstract some of the more complex C/C++ code that users would otherwise have to write to achieve the same results, into useful functions specific to the platform. It is therefore unsurprising that the novice Arduino programmers in my earlier study—all end-user developers—experienced numerous learning barriers when using it (Booth and Stumpf 2013). Equally, it may be easy to connect Arduino to a solderless breadboard, but in connecting components to the Arduino board and one another, to create a circuit, the same concepts and rules apply as in any other circuit—i.e., electrical circuit theory that so many learners struggle to understand and apply (section 2.3).

We might conclude that modular hardware platforms based on plug-and-play components will remove the need for end-user developers to acquire knowledge of electronics engineering—certainly, these platforms show promise within a rapid prototyping context (Sadler, Shluzas, and Blikstein 2017)—and that environments which simplify programming considerably, for example, visual environments, will take all of the pain out of programming physical computing devices. However, with the exception of LEGO Mindstorms, to date there is limited evidence of adoption

of these tools by end-user developers. By comparison, there is undeniable evidence of Arduino's popularity in physical computing hobbyist communities, despite it potentially requiring more specialised knowledge to use than some of the other tools. What might explain this? Perhaps these tools present different types of barriers and constraints.

Cost may be a factor. Arduino has a relatively low cost of entry—the boards are affordable, especially clones, and many of the components used to create Arduino-based circuits—with the exception of shields—are standard, off-the-shelf electronic components. By comparison, modular hardware platforms are typically more expensive. An artist wishing to create an installation involving a large number of LEDs, for example, may therefore face considerable expense if they were to use modular LEDs instead of standard ones. The form factor of modules may also physically constrain what can be developed, sometimes being larger than standard components, due to the additional circuitry they contain, or they may rely on proprietary types of connection, limiting their flexibility. However, for some end-user developers, the limitations or trade-offs of modular hardware platforms may be well-outweighed by the ease of use in rapidly developing a working prototype, without the difficulty of conventional circuit construction and its potential obstruction to creativity (Sadler, Shluzas, and Blikstein 2017).

Programming environments and notations may also impose constraints. For example, visual programming has achieved recognition, increasingly so in recent years, as a way to make programming easier for non-experts, particularly children. However, a common criticism of visual programming tools like Scratch is that while they may be useful for learning some of the basic concepts in programming, in practice their simplicity sometimes limits what is programmatically achievable. I am not aware of any empirical work establishing the boundaries of what physical computing devices can be developed using visual programming tools, and this is beyond the scope of my research, but it is worth noting.

The Arduino platform's association with the Maker Movement, from its outset, no doubt boosted its popularity, leading to a plethora of Arduino-related learning and community resources, both online and offline. Currently, Arduino is still probably the most *well-known* physical computing platform, which may also go some way to explaining why it continues to be the most widely used, despite the potential challenges it may present to novices.

Recently, alternatives beyond the 'Arduino monoculture' (Blikstein 2015) have begun to appear, in the HCI research community at least, for easier creation of physical computing prototypes

using off-the-shelf components. Taking a generative design approach, Trigger-Action-Circuits (TAC) (Anderson, Grossman, and Fitzmaurice 2017), allows novices to specify high-level behaviours using a visual programming metaphor. The system then generates candidate circuit designs for the user to choose from, according to their needs or preferences, for example, cost, component availability or ease of construction. A visual representation of each circuit is provided, for the user to copy/reproduce, along with detailed assembly instructions. In a user study, participants using TAC to develop the simplified Love-O-Meter (a popular choice for physical computing study tasks since its use in Booth et al. 2016) performed much better than those using the normal Arduino tools, both in task success and time to complete. The authors acknowledge that this kind of high-level design/development may not lead to the same degree of learning, but for end-user developers focused mainly on the end product of their efforts, it may be an effective way to take some of the pain out of developing physical computing devices. The caveat, of course, as with the other tools involving virtual circuits as a precursor to physical circuit creation, is that the user still needs to reproduce the physical circuit—my studies show that end-user developers can still experience problems and make mistakes, introducing bugs, when physically reproducing even simple circuits from images.

2.8 Supporting troubleshooting and debugging

Software engineering deals, to a great extent, with finding and fixing bugs:

"the realization came over me with full force that a good part of the remainder of my life was going to be spent in finding errors in my own programs" (Wilkes 1985)

End-user developers should not only be supported in *creating* their physical computing devices, they should be also supported in *troubleshooting* any problems which arise in process of doing so, for example, finding and fixing any bugs they introduce, or any failure of their device to behave as expected.

Troubleshooting is a form of problem solving (Jonassen 2000). In typical use, the term describes the process of locating and rectifying faults in electronic systems or circuits, in the same way that *debugging* describes the act of finding and fixing program faults that a developer has introduced. In fact, debugging has previously been referred to as an instance of troubleshooting in which program errors rather than device errors are located and corrected (Katz and Anderson

1987). In physical computing, particularly in materials aimed at end-user developers or makers (e.g. Banzi 2009), the terms *troubleshooting* and *debugging* are sometimes used interchangeably, as I do at various points in this thesis.

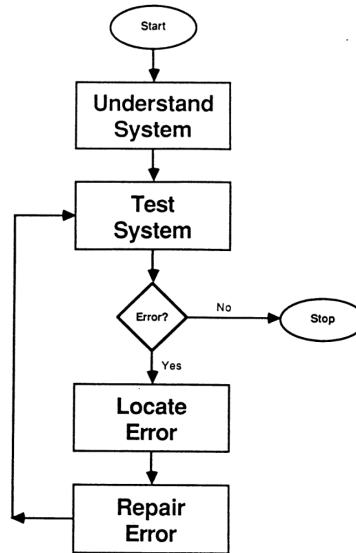


Figure 2. General model of troubleshooting. From Katz and Anderson, 1987

Troubleshooting (or debugging) is typically modelled as an iterative process, as in the Katz and Anderson *General Model of Troubleshooting* (Figure 2) (Katz and Anderson 1987)—in this model, iterations comprise 'Test System', 'Locate error', and 'Repair error' activities. During *Test System*, the developer tries to confirm the existence of a bug; in *Locate error*, they attempt to localise and identify the bug; in *Repair Error*, they fix the bug; finally, it needs to be verified that the fix has been successful, which leads back to *Test System*. If the test confirms that a bug still exists, the developer returns to localisation... and so on.

In practice, particularly for end-user developers, as I will also show in my studies, these activities sometimes co-occur, or happen in a different order, for example, end-user programmers often make a number of changes until they establish what fixes the problem or results in the desired behaviour (Ko et al. 2011), using 'repairs' to locate error—a strategy that Rosson and Carroll refer to as “debugging into existence” (Rosson and Carroll 1993).

Knowing where to look for bugs, what evidence to look for, how to test for the existence of bugs and how to interpret the result of testing, is crucial to successful troubleshooting (Morris and Rouse 1985). These are diagnostic activities in which hypotheses play a key role (Pennington and Grabowski 1990; Jonassen 2000)—troubleshooting activities are directed by ideas about the possible or probable cause of symptoms, and false assumptions made by novice programmers

and end-user programmers when hypothesising during debugging have been shown to result in new bugs (e.g., Ko and Myers 2003; Gugerty and Olson 1986). Successful troubleshooting requires a broad range of domain-specific and more generic knowledge—system knowledge, procedural knowledge and strategic knowledge, for example—which experts build up over time, forming deep and complex mental models that provide a bedrock for successful fault diagnosis (Jonassen 2000; Lesgold and Lajoie 1991). However, without the right knowledge, end-user developers—particularly less-experienced ones—may have difficulty forming hypotheses (Ko, Myers, and Aung 2004), be unable to identify appropriate strategies or procedures to test their hypotheses (Kissinger et al. 2006), or lack the practical know-how to put their ideas into action (Ko, Myers, and Aung 2004).

2.8.1 Troubleshooting software problems (debugging)

As discussed in section 2.2.3, much has been done to understand how end-user programmers debug software in order to find ways to support them, and a number of tools, informed by this work, have been developed to help end-user programmers diagnose problems, generate and test hypotheses, and localise bugs.

A popular approach for tools developed to support end-user programmers' debugging is through providing features directly within the programming environment, in order to support problem-solving activities during programming. *The Idea Garden*, an extension to end-user programming environments, scaffolds end-user developers' problem solving during programming and debugging, suggesting appropriate strategies for overcoming barriers, and providing information about useful concepts and patterns (Cao et al. 2015). This situated support tool, informed by problem-solving theory (Cao et al. 2013, referring to Simon 1980) and Minimalist theory (Carroll 1998), gently guides end-user programmers in developing their own problem-solving skills, rather than solving their problems for them. The *WYSIWYT* (What You See Is What You Test) testing methodology has proven effective in helping end-user programmers test and localise spreadsheet bugs by providing visual feedback about the “*testedness*” of cells containing formulas (Ruthruff, Burnett, and Rothermel 2005). This methodology uses a strategy the authors refer to as “Surprise-Explain-Reward”, again informed by Minimalist theory and Attention Investment theory, to reveal information useful to the user in debugging a spreadsheet. A variant of this latter tool—*WYSIWYT/ML*—adapted for machine learning environments, helps end-user programmers successfully debug problems in an intelligent agent

system (Kulesza et al. 2011). Also in the spreadsheet domain, *StratCel*, an add-in for Microsoft Excel, was designed to support a common strategy used by end-user programmers when debugging—"to-do listing". It enables end-user programmers to offload some of the cognitive effort of planning and then tracking what they test, by first automatically generating a list of things to check and then allowing the user to mark these off as they progress (Grigoreanu, Burnett, and Robertson 2010). In a study, using this tool, end-user programmers found and fixed far more bugs than those not using it, demonstrating that support tools which adopt a strategy-based approach can increase end-user programmers' success in localising and resolving their bugs. Addressing a different problem, the *Whyline* explicitly supports novice programmers' hypothesis formation during debugging, by allowing them to ask "why did" and "why didn't" questions about their programs' behaviour, and mapping these to the sections of code potentially responsible for failure (Ko and Myers 2004; 2008).

To design tools to support end-user developers in the physical computing domain, we can potentially draw much inspiration from this work, but we first need to know what they need support with. My work addresses this gap.

2.8.2 Troubleshooting physical computing problems

Until recently, there were few support tools for troubleshooting the simple circuits typically involved in end-user developers' physical computing prototype development. Most systems supporting the troubleshooting of electronics-based systems were aimed at professionals, or those training to be professionals, for example, *SHERLOCK* (Lesgold et al. 1992), an environment for teaching sophisticated electronics troubleshooting to fighter airplane engineers.

Since publication of the findings from the first study in this thesis, other researchers have developed a number of novel hardware addons or tools to help novices troubleshoot electronic circuits. Most of these tools address the lack of visibility about the internal operating states of physical computing prototypes at runtime, employ some kind of enhanced (instrumented) breadboard, with measurement data collected (automatically or on demand) from the circuit, and present this visually via a GUI, along with additional debugging features, in some cases. For example, *Toastboard*, aimed at novices and informed by interviews with domain experts, measures and visualises voltage, and can alert the user to common errors (Drew et al. 2016), while *CurrentViz* uses the virtual 'Breadboard' view in Fritzing to show calculated real-time visualisation of current flow within a breadboard circuit (Wu, Shen, et al. 2017). These tools show

promise, however, I have yet to see more formal empirical studies demonstrating their efficacy in typical end-user development tasks. Also, most only support the troubleshooting of circuits, not program-related problems.

While, as discussed in section 2.7, there are now platforms and environments which make physical computing development somewhat easier for non-experts, they usually lack the sophisticated types of debugging tools found in most professional programming environments. The current Arduino IDE has a message console, where compiler errors are displayed, as well as a Serial Monitor terminal that can display program output and which is often used for debugging, but until a recent beta version of Arduino IDE, which contains a live debugger, there was no specific debugging support available within the platform. Certain plugins enable the debugging of Arduino projects in Microsoft Visual Studio, however, as others have argued, a fully-fledged professional IDE may be daunting to inexperienced end-user developers (Repenning and Ioannidou 2006). Equally, novice end-user developers appreciate the simplicity of the Arduino's notepad-style programming interface, despite it providing them with very little assistance in troubleshooting their problems (Booth and Stumpf 2013).

Bifröst (McGrath et al. 2017) is the first tool I have found aimed at end-user developers that provides tools to help users debug or troubleshoot their programs, circuits and the interaction between these. Like some of the circuit troubleshooting support tools previously mentioned, its dashboard-style interface exposes visualisations of electrical activity (digital and analog signals) alongside an enhanced serial console and code editor, and allows users to set breakpoints, monitor variables and navigate back and forth within a recorded trace. Electrical signals are captured from the Arduino pins via a custom PCB shield, connected to a logic analyser.

To my knowledge, the circuit debugging tools I have described are still at the conceptual research prototype stage, and without further studies, beyond the exploratory work reported, there remain unanswered questions regarding their efficacy in helping end-user developers, particularly novices, resolve their problems, or whether they pose different problems. In the meantime, there is currently still little support for this population in troubleshooting. Some websites and books provide checklists for troubleshooting problems and failure (e.g. Taylor 2010; Craft 2013), but as previously discussed, end-user developers in this domain might benefit from different approaches to scaffolding their troubleshooting—the physical card-based support tool I developed, which I will describe in Chapter 5 (see also Booth et al. 2019), informed by the empirical work I will reported in the next two chapters, represents such an approach.

Chapter 3

Problems experienced by end-user developers in a physical computing task (Study 1A)

3.1 Introduction

To support end-user developers in developing physical computing devices, we need to understand what challenges they face.

In this chapter I describe a study investigating the problems faced by end-user developers when developing physical computing devices. This was an exploratory study, with the goal of addressing a lack of knowledge about end-user developers in this domain, and establishing what they have most trouble with during development. It aimed to address the following.

Firstly, the lack of empirical knowledge about the type and extent of the problems faced by end-user developers during development, and the locations or activities in which these occur. As physical computing development involves both programming *and* electronics, there is potential for different problems than have been observed in programming alone.

Secondly, knowing whether factors such as self-rated expertise and self-efficacy play a role in the problems, performance and success of end-user developers developing in this domain, should indicate whether some end-user developers are more in need of certain types of support.

Finally, knowing what types of problems end-user developers are able to resolve, and which are more likely to result in task failure, will enable us to address/target support efforts towards helping end-user developers to overcome the most difficult and severe issues.

The study seeks to answer the following thesis-level research question:

TRQ1: What problems do end-user developers experience when developing a physical computing artefact?

This has been broken down into the following three study-level research questions:

RQ1: How many problems do end-user developers encounter, and where do these problems occur? Are there aspects of physical computing development that are particularly prone to problems?

RQ2: How do end-user developers' self-rated expertise and self-efficacy affect the challenges they face in a physical computing development task?

RQ3: What problems are overcome by end-user developers, and what problems prove insurmountable?

Work that has helped to determine how end-user programmers can be supported in programming and debugging spreadsheets or web mashups is underpinned by empirical data about the difficulties that end-user programmers experience in these domains. The main part of the study was therefore an observation of the problems which arose when end-user developers undertook a hands-on physical computing development task.

3.2 Method

3.2.1 Overview

To answer the research questions, I conducted an empirical user study. Participants were given 45 minutes to develop a physical computing device from scratch to a set brief, while thinking aloud; all were given the same task specification and equipment. A rich set of data was collected: questionnaires captured information about participants' backgrounds and self-efficacy, the task was video recorded (on-screen and off-screen activity), a post-task interview captured participants' understanding of the physical computing concepts involved in the task, and any program and circuit artefacts created during the task were saved for later scrutiny.

3.2.2 Participants

Once full ethical approval for the study had been granted (Appendix A), twenty adult end-user developers were recruited. Eligible participants had to have some experience of using Arduino, but only for personal projects (for example, a hobbyist developing a microcontroller-based music-making device), or to support their own work (for example, a researcher developing an interactive prototype for use in a study they were running). They could not be (or have ever been) employed or commissioned by others specifically to create physical computing prototypes, whether or not for monetary gain.

As this was an exploratory study, I hoped for a broad mix of backgrounds and skills in electronics, programming, and physical computing across the sample. As discussed in the previous chapter (section 2.1.1), not all end-user developers in physical computing are novices. Some may have more experience in a particular aspect of development, for example, an end-user developer might be employed as a professional programmer, but dabble with Arduino in their spare time, and be relatively new to working with electronics. I wondered if any disparity in skill might reveal anything interesting—for example, whether participants who rated themselves higher in programming expertise might experience fewer programming problems than those who rated their programming expertise lower. Therefore, even professional programmers or engineers could take part, but only if they met the end-user developer criterion with regards to physical computing.

Table 1. Study 1A inclusion/exclusion criteria for participation

Inclusion criteria	Exclusion criteria
Adult (Aged 18 or older)	Aged under 18
At least <i>some</i> practical (hands-on) experience of using the Arduino platform, with, as a minimum, <i>both</i> of the following (although not necessarily in the same project): ▪ Experience of using LEDs in an Arduino project AND ▪ Experience of using at least one type of analog sensor in an Arduino project	No practical (hands-on) experience of using the Arduino platform. Experience of using <i>either</i> LEDs <i>or</i> analog sensors in an Arduino project, but not both.
End-user developer: Has only developed physical computing prototypes/devices for own use.	Previously or currently employed/commissioned specifically to develop physical computing prototypes/devices.
Able to attend, in person, a 1.5-hour session	Unable to attend, in-person, a 1.5-hour session

As a minimum, participants had to have experience of using LEDs in an Arduino project, as well as at least one type of analog sensor, although not necessarily together. This would ensure some familiarity with the types of components involved in the task, and the methods used to programmatically control them.

Table 1 summarises the inclusion and exclusion criteria for participation in the study. If a prospective participant met *all* of the inclusion criteria, they were deemed eligible to take part; a person meeting *any* of the exclusion criteria was deemed ineligible.

3.2.2.1 Recruitment

Recruitment was via several channels. As I wanted to target hobbyists, I contacted a number of different Maker communities (e.g., London Hackspace, MzTEK, London threads-space, London Arduino meetup group, Not Just Arduino group, and Dorkbot London, and with permission, sent a call for participation to their mailing lists. To reach end-user developers in other communities, makers in my personal network disseminated my call for participation online and offline, including via several university student mailing lists, and a poster pinned to noticeboards at several London universities (Appendix B). I also posted on social media at regular intervals (Twitter and Facebook), first setting up a Google document containing basic information about the study that I could link to, keeping emails and posts succinct.

Interested respondents were sent a copy of the participant information sheet (Appendix C) providing full details about the study, so that they could determine their eligibility and decide whether they wished to participate. Prospective participants were screened for eligibility and only those who met all of the criteria for participation were invited to take part.

3.2.2.2 Who took part?

Twenty adult end-user developers were recruited—8 females and 12 males, with a mean age of 31.8 years. Table 2 shows the ages, gender, and occupations of those who took part. Demographic, experience and expertise data were collected from participants via a background questionnaire (Appendix E, described in sections 3.2.3.1 and 3.2.5.1); self-efficacy ratings were captured via a second questionnaire (Appendix F, described in sections 3.2.3.3 and 3.2.5.2).

Descriptive statistics were used to summarise these data (Appendix V). As hoped, there was a broad mix of backgrounds and experience across the sample, which I will now describe in greater detail.

Occupation

Participants included both students (6) and professionals (14) and came from a variety of disciplines. Only two people *currently* employed as IT professionals took part—one software developer and one systems analyst—but seven participants indicated that they had been employed as programmers *at some point*. Only one engineering professional—a broadcast engineer—took part. All were end-user developers in physical computing, as specified.

Table 2. Study 1A participants

Ptc	Age	Gender	Occupation
P01	27	Female	Post-Doctoral Researcher (Human-Computer Interaction)
P02	27	Male	Broadcast Engineer
P03	22	Female	PhD Student (Computer Science)
P04	25	Female	PhD Student (Media & Arts Technology)
P05	32	Female	Project Manager (Arts)
P06	46	Male	Events/Content Producer
P07	30	Male	PhD Student (Media & Arts Technology)
P08	33	Male	Restaurant owner
P09	29	Female	Director and Research Consultant (Technology & Arts)
P10	34	Female	Project Manager (Media & Technology)
P11	53	Male	High School Substitute Teacher (English Literature)
P12	41	Female	University Lecturer (Fashion Marketing)
P13	38	Female	Student (Science & Human Physiology)
P14	32	Male	Software Developer
P15	32	Male	Post-Doctoral Researcher (Computer Science)
P16	29	Male	Systems Analyst
P17	28	Male	PhD Student (Human-Computer Interaction)
P18	30	Male	Education Programme Manager (Science)
P19	26	Male	Industrial & Web Designer
P20	22	Male	MSc Student (Computer Science & Embedded Systems)

Experience

As shown in Figure 3, participants generally had more years of programming experience (mean=10.89, SD=7.53) than electronics experience (mean=6.75, SD=7.63) or physical computing development experience (mean=3.23, SD=2.03).

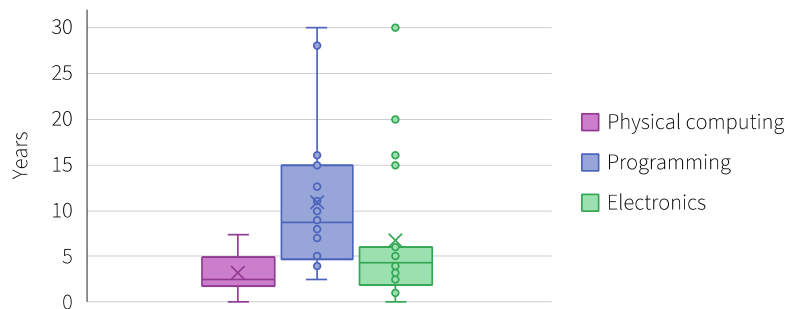


Figure 3. Participants' experience, in years

Training / instruction

Most (16 participants) had received some form of training or instruction in programming (Figure 4), ranging from one-off workshops to formal education—twelve had attended at least one programming module at a university level. Fewer had received training in electronics (9) or physical computing (13). Most physical computing instruction had been in the form of workshops, often introductory, although three participants had taken a university module. Instruction in electronics was least represented. Five had taken at least one module at university level, while four had been taught electronics at school or attended specific workshops.

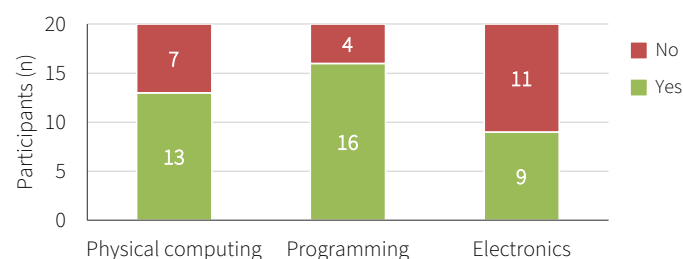


Figure 4. Participants' training

Self-rated expertise

To succeed in a physical computing task, end-user developers need to be sufficiently proficient at programming, and at constructing an electronic circuit, but we would hardly expect them to be experts. Participants rated their expertise in programming, electronics and physical

computing on a scale of 1 (complete beginner) to 7 (total expert). On average, rated themselves slightly higher in programming expertise (mean=4.40, SD=1.47) than electronics (mean=3.10, SD=1.33) or physical computing (mean=3.60, SD=1.19) (Figure 5). These are then not all complete novices but represent a good cross-section of end-user developers in physical computing.

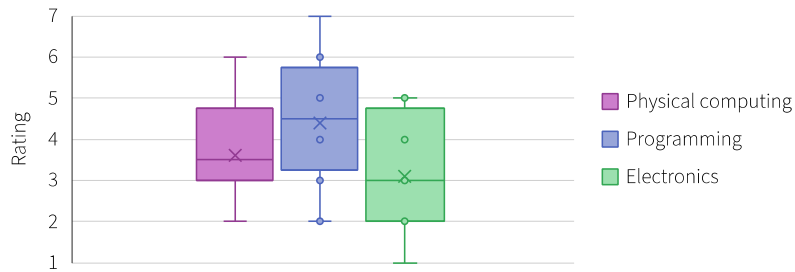


Figure 5. Participants' self-ratings of expertise in physical computing, programming and electronics

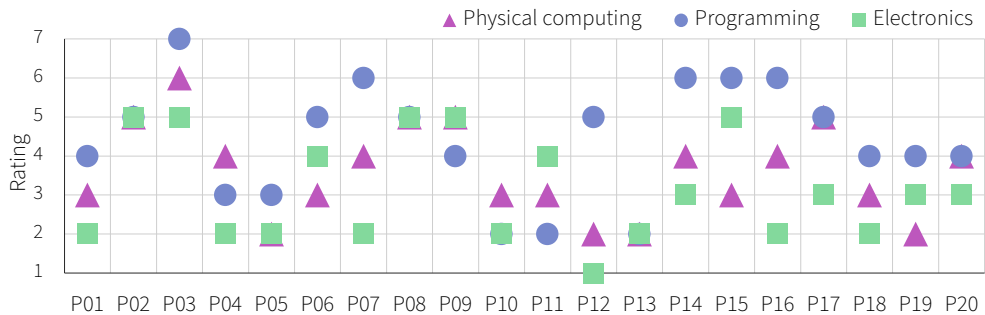


Figure 6. Individual participants' self-ratings of expertise (1-7)

Self-efficacy

Self-efficacy is an individual's belief in their capability to accomplish something, even in the face of difficulty. Participants' self-efficacy was captured via a separate questionnaire (Appendix F, described in sections 3.2.3.3 and 3.2.5.2), with analysis resulting in a single self-efficacy score per participant, out of a maximum of 100. As shown in Figure 7, most participants were relatively self-confident at tackling a physical computing task of moderate complexity using the Arduino platform, with a mean score of 69.70 for the sample (SD=10.78).

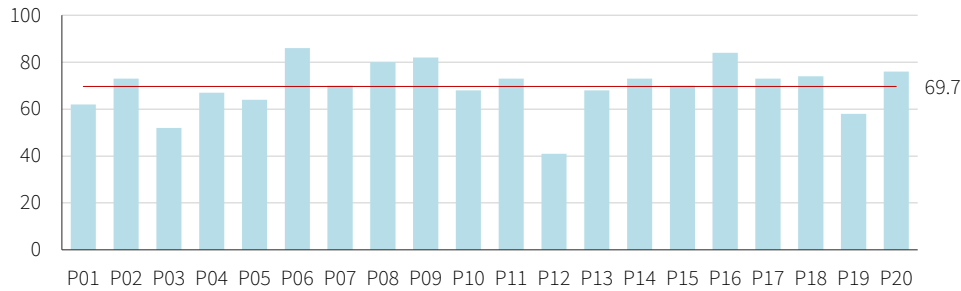


Figure 7. Individual participants' self-efficacy scores (out of a maximum of 100)

3.2.3 Materials

A variety of materials were developed for this study. These will now be described.

3.2.3.1 Background questionnaire

This questionnaire (Appendix E) captured demographic information from each participant, as well as information about their background and experience in programming, electronics and physical computing. This was created using online survey software (SurveyGizmo, now known as Alchemer (Alchemer LLC, n.d.)) and was completed by participants in advance of the in-person lab session, to keep the time required for in-person participation to a minimum (see section 3.2.4.1). An initial statement asked participants to indicate their consent to this data collection, prior to answering any questions.

3.2.3.2 Informed consent form

An informed consent form (Appendix D) was created to capture each participant's agreement to taking part in the study and their acknowledgement that they understood what this would entail, including video recording of the session, and the uses of any data gathered.

3.2.3.3 Self-efficacy questionnaire

The self-efficacy questionnaire (Appendix F) was based on a validated, standard questionnaire about self-efficacy in computer use (Compeau and Higgins 1995), and captured participants' self-efficacy in completing a physical computing task of moderate complexity using the Arduino platform.

3.2.3.4 Physical computing development task

The main part of the in-person session was the hands-on physical computing development task. For this task, participants were asked to develop a physical computing device, from scratch, to a given brief, using the Arduino platform.

The task was designed to involve some of the most fundamental concepts in physical computing. It was a simplified³ version of the third project in the official Arduino Starter Kit ('Arduino Starter Kit' n.d.), which is aimed at people new to the Arduino platform. As this is an early project in the set of ten Starter Kit project tutorials, it was reasonable to assume that participants would already have some exposure to most, if not all, of the concepts involved in it, for example, connecting LEDs to an Arduino and programmatically controlling them, or connecting a sensor to an Arduino and programmatically reading data from it.

The device that participants were asked to build was a "Love-O-Meter", which uses three LEDs to visualise the readings from a temperature sensor (Figure 8). The successfully constructed device should behave as follows: When the sensor measures the ambient room temperature, no LEDs should be lit. As temperature increases, LEDs should light up in turn, as specific temperature thresholds are reached, until all 3 LEDs are lit. As the temperature drops, the LEDs should turn off, one by one, as the same temperature thresholds are reached, until the ambient room temperature is reached, and no LEDs are lit. Temperature is changed by holding the sensor between the fingers (to increase the temperature) or releasing it (to decrease the temperature).

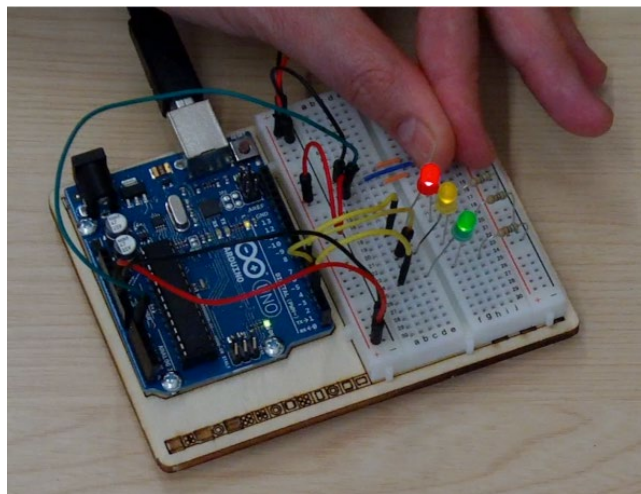


Figure 8. Love-O-Meter prototype in action

Building the Love-O-Meter device requires creating an electronic circuit and programming it to achieve the specified behaviour. It is possible to first build the whole circuit and then write the whole program, or alternatively, to decompose the task into smaller parts, building the circuit

³ The original project included code that converted the raw ADC (analog to digital conversion) readings (read from the analog pin), first to voltage and then Celsius. The specification used in this study included no conversion, reducing the complexity of the program.

and writing the program for discrete functional units (e.g., the sensor circuit and code), and establishing that they work as expected before extending or combining them. Indeed, such an incremental approach to development is considered good practice, however, to give an idea of what is involved in constructing the circuit and writing the program, I will here describe them as two separate activities:

Building the circuit

Building the circuit requires seating 7 electronic components (1 x temperature sensor, 3 x LEDs and 3 x resistors) in a solderless breadboard and connecting them to appropriate pins or sockets on the Arduino microcontroller board, and to one another, where appropriate. Figure 9 shows an annotated layout of the simplest circuit that could be built to satisfy the task brief.

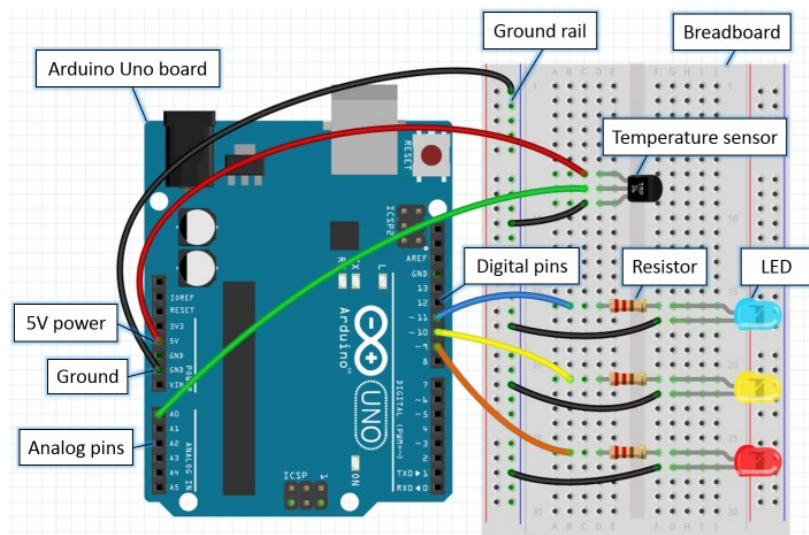


Figure 9. Model Love-O-Meter circuit

The temperature sensor is an analog device, therefore it must be connected to an analog pin on the Arduino board, so that continuous temperature values can read from it. At runtime, the Arduino will convert these analog values (voltage readings) into digital values (in the range 0-1023) that can be used by the program. The sensor has 3 legs, and each requires a specific type of connection. Although more complex configurations can be used, the sensor requires, at the very least, a connection to Ground, a connection to power (e.g., the 5V pin on the Arduino) and the connection to an analog pin already mentioned. Connecting the sensor legs incorrectly (e.g., swapping two of the connection types) can result in erroneous behaviour, or even damage to the sensor.

The LED anodes must be connected to digital pins on the Arduino board, so that these pins can be used as switches, by the program, to turn the LEDs on or off. The LED cathodes must be connected to Ground. LEDs are diodes, which means they have polarity—they work only in a specific direction. Therefore, it is crucial that these connections are not reversed, or the LEDs will not work. A resistor should be connected in series with each LED, either between the cathode and Ground, or between the anode and the digital pin, to regulate current. Not doing so can lead to burning out the LEDs prematurely or even damaging the Arduino. The resistor needs to be of a value appropriate for the LED, in the context of the circuit. Too low a value and the resistor will not prevent damage; too high and the LED will not light up.

As the sensor and LED components require, in total, four connections to Ground, but the Arduino board only has three Ground pins, it is necessary to create a Ground rail on the breadboard, which allows multiple components to share a common Ground connection.

Finally, connecting the circuit, via the USB port on the Arduino board, to a USB port on the computer, enables communication between the IDE and the Arduino board, so that information can be sent back and forth between them, for example to upload the program to the microcontroller or to return values from the microcontroller to the IDE. This connection also, importantly, provides power to the Arduino board and thereby the rest of the circuit.

Writing the program

Creating the program requires writing code that reads the current temperature from the temperature sensor, compares this value to specified temperature thresholds, and then turns the LEDs on or off, as appropriate. Figure 10 shows what the program might look like.

An Arduino program always contains two specific functions: the `Setup()` function, which is run once when the Arduino is started, and the `Loop()` function, which runs repeatedly in a loop once the `Setup()` function has been processed and contains the main body of the program.

Pins on the Arduino to which the sensor and LEDs are connected must be specified, in code, as input or output, in the `Setup()` function. A `Serial.begin()` statement must also be added to the `Setup()` function, to enable serial communication between the Arduino and the IDE. Variables can be created to store the temperature values read. In the main body of the program—the `Loop()` function—the temperature is read by using the `analogRead()` function, which reads the current value of the sensor pin. The main logic of the program, i.e., the

code which compares the current temperature value to specified threshold values and turns LEDs on or off, as appropriate, is implemented in the form of conditional statements.

```
int led1 = 9; //declare variables for the LEDs and set them to digital pin numbers 9, 11 and 13
int led2 = 11;
int led3 = 13;

int tempSensor = A0; //declare a variable for the temperature sensor, and set it to analog pin A0
int temperature;      //declare a variable to hold the reading from the temperature sensor

void setup() { //this will run once, when the program starts

  pinMode(led1, OUTPUT); //set the digital pin (9) connected to LED#1 to be OUTPUT
  pinMode(led2, OUTPUT);
  pinMode(led3, OUTPUT);
  pinMode(tempSensor, INPUT); //set the analog pin connected to the sensor to be INPUT

  Serial.begin(9600); //start serial communication between the Arduino and the computer/IDE
}

void loop() { //this will run repeatedly, once the setup() function has completed

  temperature = analogRead(tempSensor); //read the sensor pin and assign the value to the 'temperature' variable
  Serial.println(temperature); //write the temperature value to the Serial Monitor window in the IDE

  if (temperature >=149) { //if the sensor reading is 149 or greater...
    digitalWrite(led1, HIGH); //turn the first LED on
  } else { //if the sensor reading is less than 149
    digitalWrite(led1, LOW); //turn the first LED off
  }

  if (temperature >=152) {
    digitalWrite(led2, HIGH);
  } else {
    digitalWrite(led2, LOW);
  }

  if (temperature >=155) {
    digitalWrite(led3, HIGH);
  } else {
    digitalWrite(led3, LOW);
  }

  delay(200); // wait 200 milliseconds before the next reading
}
```

Figure 10. Model Love-O-Meter program

To determine appropriate values for the temperature thresholds, it is necessary for the participant to find out what values are being read from the temperature sensor. This is done by using the `Serial.println()` function to output values read from the sensor to the IDE's Serial Monitor—a terminal window that can be launched within the IDE, often used for debugging. Observing the values output to this window, will enable the participant to identify first, the value for the ambient room temperature (when the sensor is not held) and second, the rate of increase in values when the sensor is held, so that threshold values can be chosen for use in the main logic of the program, that is, appropriate values at which to turn the LEDs on or off.

LEDs are turned on/off with the `digitalWrite()` function, using the argument `HIGH` to turn an LED on, and `LOW` to turn an LED off. The `delay()` function is used to pause for a specified number of milliseconds between sensor readings.

Task resources

A task instruction sheet (Appendix G) was created, specifying the prototype requirements, including desired behaviour and details of any constraints within which the participant was expected to operate, for example, the task time, and rules for use of external resources.

The development board chosen for use in the task was the official Arduino UNO revision 3. This is a commonly used starter board, included in the official Arduino Starter Kit. The development environment used was the official Arduino IDE (version 1.61 for Windows), running on a Microsoft Windows 7 desktop PC. The PC had access to the internet and the Chrome browser was available for browsing the Web.

A kit of equipment was created, for use in the task, starting with a selection of electronic components from the Arduino Starter Kit. This included TMP36 temperature sensors, different coloured LEDs (red, green, yellow, blue) and resistors in a wide range of different values (4.7Ω, 220Ω, 330Ω, 560Ω, 1kΩ, 10kΩ, 1MΩ, 10MΩ). Several of each component were provided, in case participants wanted (or needed) to swap out components during the task, and only new components were provided, so that there was no chance of a participant using a component that had been damaged by previous use, or choosing a particular resistor because it had been used by a previous participant (used resistors are obvious in that they are no longer perfectly straight). The components were labelled and only component types needed for the task were provided, as I was more interested in whether participants could develop the prototype, than in whether they were able to recognise components. Several values of resistor were provided as this would allow me to see whether participants were able to select appropriate values of resistor for use with the LEDs. The kit also contained jumper wires (also known as jump wires), which are a type of wire designed for prototyping convenience—they terminate in a hard pin at each end, making it easy to insert them into breadboard holes or the pins on an Arduino board. Several different colours of wire were provided, to enable participants to follow wiring colour conventions (e.g., red for power, black for ground, different colours for different signals/components etc) should they wish to do so. The kit also contained a USB cable, for

connecting the Arduino board to the PC, and a digital multi-meter, in case participants wished to measure any aspect of their circuit.

3.2.3.5 Interview topic and questions guide

A guide was created to direct questioning in the interview that followed the task, to elicit participants' understanding of physical computing concepts. Topics and questions related to the concepts and equipment involved in the task. To create it, a range of resources were consulted, including books about physical computing and Arduino, websites about physical computing development, online physical computing tutorials, and literature on programming, electronics and physical computing. Although captured, interview data were not analysed—I originally planned to also look at mental models as part of this investigation into end-user developers' problems, but decided instead to investigate troubleshooting, in depth, (Chapter 4), with that ultimately becoming the focus of this PhD (discussed further in section 7.3).

3.2.4 Procedure

Once it was established that a respondent was eligible and wished to take part in the study, a date and time were agreed for them to attend an in-person session in the Centre for HCI Design's Interaction Lab at City, University of London. They were then sent a confirmation email containing instructions for attending the session, and a link to the background questionnaire, to be completed prior to the session.

The participation procedure is summarised as follows:

1. The participant first completed a **background questionnaire** online, at an emailed URL.
2. They subsequently attended a scheduled **in-person session**, structured as follows:
 - 2.1. Completion of a **self-efficacy questionnaire** to measure their confidence in physical computing development tasks.
 - 2.2. A **hands-on task**, in which they developed a physical computing prototype, from scratch, to a given brief, using equipment provided, while thinking aloud. At the end of the task, they demonstrated what they had built.
 - 2.3. A **post-task interview**, in which they explained the prototype workings, and answered questions about the concepts involved in the task.

Figure 11 shows the sequence of activities, which will now be described in detail.

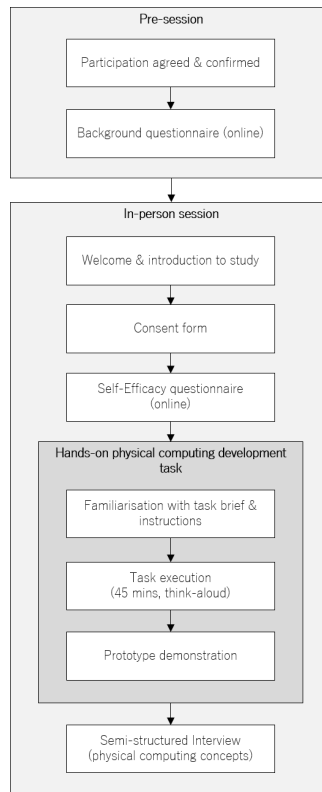


Figure 11. Study 1A: Sequence of activities

3.2.4.1 Background questionnaire (online completion)

Participants were asked to complete the background questionnaire online, in advance of the session, as the in-person session would already take a considerable amount of time (1.5 hours). A link to the questionnaire was sent in the participation confirmation email. On being opened, the questionnaire required participants to digitally provide their consent for this data collection prior to answering any questions.

3.2.4.2 In-person session

On arrival at the Interaction Lab the participant was given a verbal introduction to the study, asked to read the participant information sheet again, and signed the informed consent form.

Self-efficacy questionnaire: Participants completed the self-efficacy questionnaire, after first being familiarised with the content, including clarification of what was meant by “a task of moderate complexity”—an Arduino development task that involved prototyping a circuit which

involved a sensor and three LEDs, and writing a program to coordinate and control their behaviour.

Physical computing development task: Participants were then given the physical computing development task. They were first taken through the task information sheet verbally, ensuring that they understood the specification of the device they were being asked to develop. They were shown the equipment they could use, and the constraints within which they were expected to operate were explained. Participants were given 45 minutes to complete the task and asked to think-aloud while doing so. Should they forget to think aloud during the task, they were reminded to do so. During the task they could ask questions to clarify the brief but could expect no other help or advice in the development of the prototype. Participants were also allowed to use additional resources to help them complete the task, for example, they could use help content and example programs built into the Arduino IDE, or search online for other sources of information. They were allowed to copy code and use resources to guide the construction of the circuit, as reusing and adapting code or other content created by others is a common behaviour for end-user developers, but could not search for and copy an *exact solution* to the brief, i.e., a project in which LEDs are controlled in response to readings from a temperature sensor. Finally, they were asked to keep the prototype within a marked area of the desk during development (to keep it within the video recording frame).

When the participant was happy that they understood what they were being asked to do, the task commenced. Beyond reminders to think aloud, the only facilitator intervention was if there was any danger of harm to the participant, or if any problem with the equipment was observed that was unrelated to anything the participant had done (for example, with the PC hardware or the software running on it). Towards the end of the 45 minutes, they were given a verbal reminder of the time remaining.

At the end of the task, participants were asked to demonstrate the final behaviour of their prototype. If they had not succeeded in constructing and programming a physical prototype that met the given specification, they were asked to show how far they had got.

Post-task interview: Participants were first asked to give an overview of what they had created, explaining what they had used, how the program and circuit were structured and how the individual aspects worked together as a whole. They were then asked questions about the concepts involved in the task. Note, again, that these data were not analysed.

3.2.4.3 Pilot

The Starter Kit estimated 30 minutes as the expected time to complete the Love-O-Meter project, however, that would obviously be with the benefit of the step-by-step instructions in the manual, whereas I wanted to give participants a broad specification of what the prototype should achieve and see how they fared without further instructions. I trialled the task and on the basis of the results, increased the time to 45 minutes. The entire procedure and materials were piloted with an end-user developer who had experience of using Arduino. As a result of this, some questions in the background questionnaire (relating to experience of alternative physical computing platforms) were removed, and some questions were reworded for clarity.

3.2.5 Data collection

3.2.5.1 Background questionnaire data

The Background Questionnaire (Appendix E) contained 21 questions, capturing two types of data: firstly, personal and demographic information (age, gender, occupation) and secondly, information about participants' backgrounds and experience in *physical computing*, *programming* and *electronics*. The background and experience questions were grouped by these three activities and ordered so that participants were first exposed to the questions about physical computing, and then programming and electronics.

Length of experience: Participants were asked to indicate, in years and months, how long they had been doing each activity, from the approximate date at which they had first started. This is not an accurate measure of the amount of experience a participant has—for example, they may not have programmed frequently or even regularly since they first started programming— but it provides an idea of how long they had been exposed to the concepts involved in each activity.

Level of expertise (self-rated): Participants rated their expertise in each activity, using a scale from 1 (complete beginner) to 7 (total expert). While we cannot take this as an objective measure of their expertise, it provides some indication of the level of knowledge and skill they felt themselves to have in each activity, enabling me to look for any correlation with performance or problems (RQ2).

Knowledge acquisition: Participants were asked to indicate how they felt they had acquired their knowledge and skills in each activity. This was in a multiple-choice question, containing 5 statements that ranged from ‘totally self-taught’ to ‘totally through training and instruction’.

Training and instruction: Participants were also asked whether they had received training or instruction in each activity (Yes/No). If yes, they were asked to provide a brief summary of the training or instruction.

Employment: Participants were also asked whether they had ever been employed as a professional programmer (Yes/No) or to construct/troubleshoot electronic circuits (Yes/No).

3.2.5.2 Self-efficacy questionnaire data

The self-efficacy questionnaire (Appendix F) was based on a validated questionnaire about self-efficacy in computer use (Compeau and Higgins 1995), that has been used in a number of end-user programming studies. The word of the questionnaire was adapted for the context of the study. It contained a main statement: “*I could complete a physical prototyping task of moderate complexity using the Arduino platform...*” followed by 10 questions that related to completing this task under particular circumstances, for example “*...if I had only the built-in help facility for assistance*”. When familiarising each participant with the questionnaire, “a task of moderate complexity” was verbally defined as a task that involved prototyping a circuit that involved a sensor and three LEDs and writing a program to coordinate and control their behaviour.

In each question participants were asked to first indicate whether they thought they would be able to complete the task under the given circumstance (Yes/No). If yes, they had to rate their self-confidence in completing it on a scale of 1 to 10. The scores for these 10 questions were later summed to create a self-efficacy score out of 100, as is customary for this questionnaire.

3.2.5.3 Task data

A variety of data were collected in the hands-on task, chiefly video recordings and the prototype artefacts created by participants (program, photographs and Fritzing images of the final state of each participant’s physical circuit), and additionally, any notes made by myself or participants during the task, and participants’ browser history.

Video recordings

Data about participants' thoughts and behaviour during the task was captured through video recordings from four different vantage points, recording both on-screen and off-screen activity. Morae Recorder software was used to record on-screen activity, as well as video of the participant's head and shoulders, using the computer's webcam, so that their facial expressions should be visible. A standalone camera faced the desk from a short distance away, recording the participant's body movement, including interaction with the equipment, and their movement between the computer and the rest of the equipment. A second standalone camera was positioned directly above the desk, framing a zoomed-in, overhead view of the physical prototype (circuit). A roughly A4-sized area of the desk was marked using black tape, and the participant was asked to keep the prototype inside this marked area, to ensure it stayed within the recording frame. Audio tracks were recorded for all videos, so that these could later be used to synchronise the videos—clapping hands together before the task began created an obvious spike in the waveform of each audio track, similar to how clapperboards are used in filmmaking to facilitate synchronisation of multiple recordings.

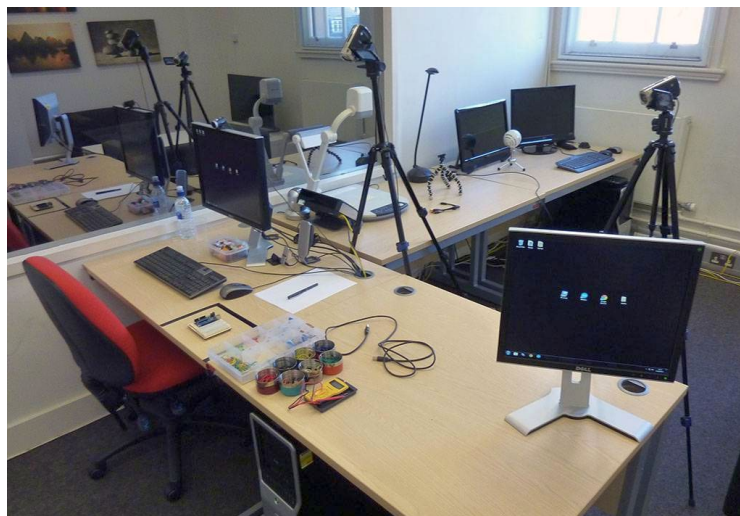


Figure 12. Desk and equipment setup. An additional monitor (visible on the right-hand side of the image) mirrored the participant's screen, enabling me to observe on-screen activity during the task in an unobtrusive way.

In preparation for analysis, all videos recorded for each participant were later synchronised, using Adobe Premiere Pro video editing software, to a single, composite, split-screen video, showing all camera views at once. Figure 13 shows a still image from one of these videos. During analysis, these composite videos enabled me to see what was happening from multiple viewpoints simultaneously.

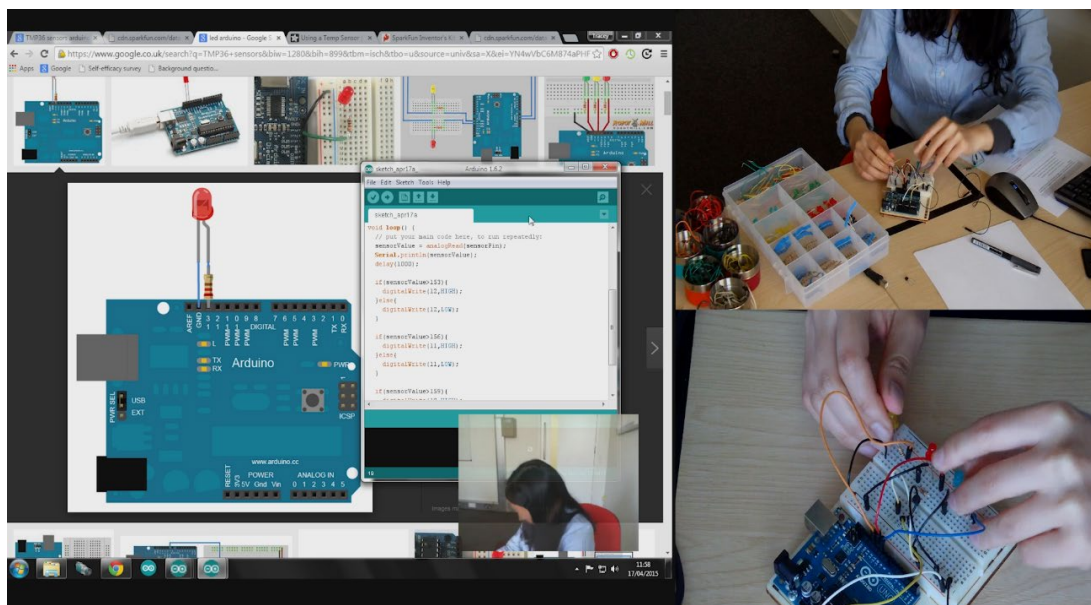


Figure 13. Still from composite split-screen video of a participant undertaking the task, showing 1) on-screen activity (large panel), 2) view of the participant's head and shoulders (small panel embedded within the screen activity panel), 3) desk-facing view (top right panel) and 4) overhead, zoomed-in view of the circuit (bottom right panel)

The content of these videos was transcribed, using Inqscribe multimedia transcription software (Inquirium, LLC, n.d.), creating a written record of what each participant *said* and *did* during the task, i.e., verbal and non-verbal behaviour. Quotation marks were used to indicate the start and end of verbatim quotes, to differentiate these from descriptions of behaviour. Creating these transcripts enabled me to immerse myself in the data, through repeatedly watching and re-watching the videos, in order to capture the verbal protocol of what participants said during the tasks, and to observe and describe what else was happening. When describing actions or other non-verbal behaviour, the aim was not to create a full, comprehensive record of all activity at the most granular level possible—pragmatically, with approximately fifteen hours of task video this would have been impractical—but to summarise participants' actions, for example “*Removes the red LED from the breadboard*”.

These written records were imported into Excel, creating twenty transcript spreadsheets in total—one for each participant—all stored within one Excel workbook. Figure 14 shows an extract from one of these spreadsheets. When analysing task data—for example, to identify problems encountered by participants (section 3.2.6.2)—the transcript spreadsheets were used in conjunction with the composite videos—I not only read the written transcripts and applied codes to these, I also scrutinised the videos when doing so, as rich representations of activity.

While selecting a resistor "I don't remember quite well what kind of resistance I should use"
"Let's go with 10" (he selects a 10k resistor)
"I can never remember the right side for the LED"
Connects the LED to the power rail and the 10k resistor, which is connected to the ground rail. The LED doesn't come on.
"I think it's a bit too high". Removes the 10k resistor and replaces it with a 330 ohm resistor.
The LED comes on. "That's okay".
Removes the resistor. The LED is now only connected to 5V (via the anode)
Takes a temperature sensor and looks at it." I can look it up on the Internet to see how it works, right?"

Figure 14. Extract from a transcript spreadsheet

Prototype artefacts: program and circuit

After each session, any programs created by the participant were saved, and the participant's circuit (the *physical* hardware part of the prototype, comprising the Arduino board, the solderless breadboard, and any components and wires that had been added to them) was digitally photographed from a number of angles. Figure 15 shows two examples of these photographs, taken of the same circuit.

The Fritzing software application was also used to create a visual record of the final circuit configuration—as the Arduino boards and breadboards were reused in subsequent participant sessions. Figure 16 shows an example of a Fritzing image of a participant's circuit.

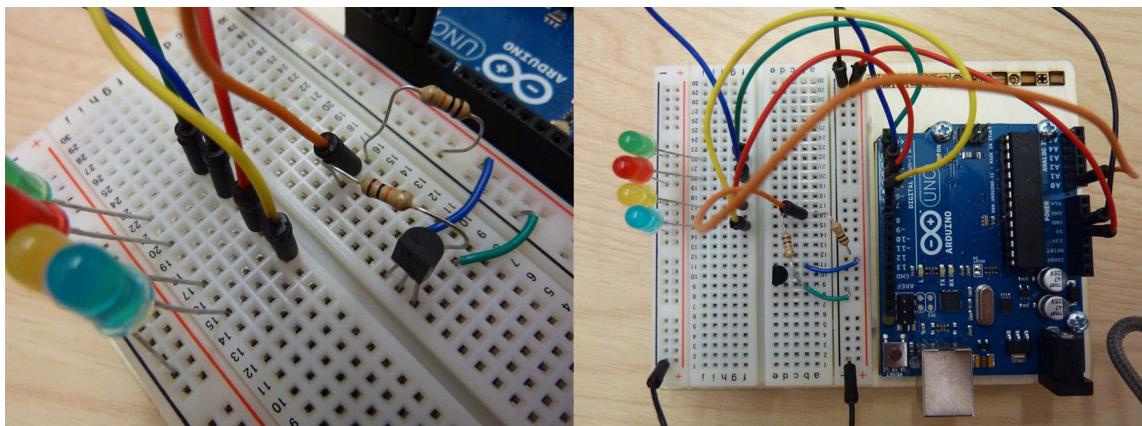


Figure 15. Photographs of a participant's circuit

Additional data

A Chrome browser plugin was used to export the participant's browser history to a separate file, to keep a record of all web pages that the participant had visited during the task, should these data prove useful during analysis. Any notes or diagrams created on paper by the participant

during the task or interview were scanned and saved as digital files, in case these also proved useful during analysis. During the task I sat close enough to the participant to observe them in an unobtrusive way (see Figure 12) and took handwritten notes—these were also scanned and saved as digital files.

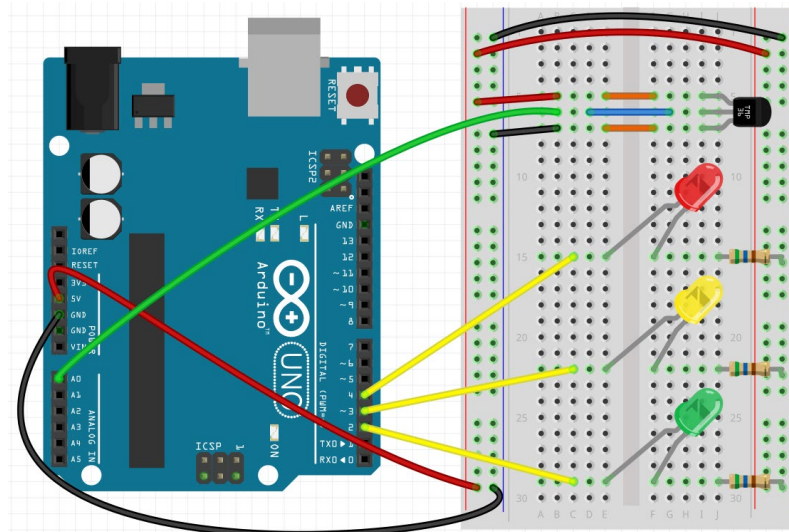


Figure 16. Fritzing image of a participant's circuit, showing the Arduino board (left) and the solderless breadboard (right)

3.2.5.4 Post-task interview data

Participants' answers to the post-task interview questions were captured as video-recordings, using the same camera angles, so that I could see not only the participant's face (and therefore their expressions) but also any gestures they made, for example, when referring to their prototype. When discussing their program, participants were asked to use a mouse, rather than their fingers, to point to things on-screen, so that this would be captured by Morae Recorder.

3.2.6 Data analysis

To answer the research questions, I analysed the data using mixed methods, summarised as follows:

RQ1: How many problems do end-user developers encounter, and where do these problems occur? Are there aspects of physical computing development that are particularly prone to problems?

To address RQ1, the task transcripts and videos were analysed for evidence of problems evident in participants' comments and behaviour (section 3.2.6.2). To determine the types of problems

experienced or introduced by participants, *problem type* codes were applied to the transcripts for each problem instance, (section 3.2.6.3). To determine where—i.e., in what aspects of development—participants experienced problems, *location* codes were applied, as sub-codes, to each of the problem type instances (section 3.2.6.4).

These results were aggregated and summarised using descriptive statistics.

The task data were also analysed for participants' task success (section 3.2.6.6). I then looked for correlation between task success and the number of problems, problem types and locations of problems. I used statistical tests to check the results for significance.

RQ2: How do end-user developers' self-rated expertise and self-efficacy affect the challenges they face in a physical computing development task?

RQ2 was answered by analysing data from the background questionnaire and self-efficacy questionnaire data in conjunction with the coded datasets from answering RQ1.

To identify any relationships between participants' backgrounds and the problems they encountered, SPSS was used to check for correlation between background variables (self-efficacy scores and self-rated expertise in programming, electronics, and physical computing) and the number of problems, problem types and problems per location. I also looked for correlation between these background factors and task success. Statistical tests were performed to check these results for significance.

RQ3: What problems are overcome by end-user developers, and what problems prove insurmountable?

RQ3 was addressed first by analysing the problem datasets (RQ1), now coding whether or not participants had managed to resolve/overcome each of the obstacles or bugs they had encountered (section 3.2.6.5).

The resulting, coded dataset was analysed for the number and proportion of problems that participants did or did not overcome, individually and across the sample, and the types and locations of these problems. Further analysis included comparing the number and proportion of each type of problem resolved in each location.

These results were analysed in conjunction with task success data (RQ1), to determine and compare how many (and what proportion of) problems, of what types and in which locations,

successful and unsuccessful participants were able to overcome. These results were further analysed in relation to program and circuit correctness (section 3.2.6.7), to check for any difference in performance between participants who failed the task but still managed to complete the program or circuit correctly, and those whose circuits or programs were incorrect.

The task transcripts of unsuccessful participants were then used to determine the cause of each unsuccessful participant's task failure (section 3.2.6.8). These results were then aggregated, to reveal what type (locations) of bugs were responsible for task failure.

Finally, once it was established which bugs participants had struggled to overcome, and which bugs had led to most task failures, I was able to look deeper into the transcripts, to get a sense of participants' difficulties in resolving these bugs.

3.2.6.1 About the coding process

In the above summary of analysis steps, and the descriptions of coding schemes that follow this section, the process of coding qualitative task data sounds far more linear than it was in practice. Coding occurred in roughly the order described, however, the coding schemes were developed and refined through considerable iteration over the dataset.

In preparation for coding, I created an initial definition of 'problem', based on looking for evidence of difficulty or impediment to progress (see section 3.2.6.2). Creating the written transcripts had already made me aware of some of the difficulties or issues that I could expect to find in the dataset, but I also compiled a list of additional things to look out for, synthesized from the literature, my previous experience of coding learning barriers (Booth and Stumpf 2013), and my domain knowledge of physical computing and of development in general. Guided by these sensitizing concepts, I read closely through the transcripts, using the videos for extra context/detail, looking for any evidence of difficulty, or impediment to progress, and applying the problem code where appropriate, making note of any new rules. As I progressed through the dataset, encountering evidence of new and different challenges faced by participants, the coding rules and list of examples evolved. If unsure of something, it was flagged up for discussion. Transcripts were repeatedly revisited, to review in light of new rules, checking previous coding for consistency, and making amendments where appropriate. While inter-rater reliability checks were not formally conducted, portions of the dataset were discussed and

jointly coded with one of my PhD supervisors on more than one occasion, which helped to ensure validity of the codes and reliability of application.

The pattern of iteration, rule refinement and revision applied to all of the qualitative analysis in this study, with frequently cycling within and between different levels of coding. For example, although coding began with problems, then progressed to problem types and locations, in practice, when coding the latter, I might re-encounter something which I felt, on reflection, should be coded as a problem, or become aware of a bug that had been somehow overlooked during the previous rounds. Notes made when coding instances of problems, in separate columns alongside the coded text, formed the basis from which the problem type and location codes developed—semi-inductively, as I was already sensitized to the concepts through familiarity with the literature. Final codes—and rules for application—were reviewed within my supervisory team before final application and extensive re-checking. This happened at two key points: firstly, when finalising the coding required to answer RQ1, and similarly, for RQ3, when the coded dataset was re-analysed, for evidence of problem resolution.

As already mentioned, codes were applied to the task transcripts, but I also used the task videos throughout the coding process. The videos were a crucial source of rich data when coding, particularly in respect to external help seeking by participants, or changes to the program or physical circuit, which often required very careful scrutiny, involving repeated rewatching, and/or magnification of the video image. However, to verify certain problems—or their resolution—I sometimes even reproduced participants' prototypes (circuit, program or both), to correspond with what I saw in the video. During the coding process, descriptions of changes to the circuit or program were amended in the transcripts where I felt this to be useful or important, for example, to add more granular details about a particular bug or bug fix.

I will now describe the coding schemes, including the rules for application, in more detail.

3.2.6.2 Problems

The task transcripts were coded for evidence of *problems*. A problem was defined as *any impediment to progress*, that is, any difficulty, action or thought that halted progress or slowed it down, or had the potential to do so. In effect, anything that *was not* something an idealised maker would do—one who knew exactly how to complete the task, without assistance, and understood every aspect of it—was counted as a problem. Inspired by research into Learning

Barriers (Ko, Myers, and Aung 2004), this definition included any verbal or non-verbal evidence that the participant was having difficulty determining how to proceed, or in assessing or understanding something—similar to what others have also termed information gaps (Kissinger et al. 2006). Inspired by previous work on error classification (Reason 1990; Ko and Myers 2005), the definition included errors made in conscious planning, assessment and action, rule-based mistakes (where expertise failed), as well as knowledge-based mistakes (where expertise was lacking). It also included slips and lapses—unconscious errors in execution. I did not apply these classifications when coding, but they were catered for within the coding rules and examples.

Types of evidence

I looked for evidence of problems in the actions, verbal comments and other non-verbal behaviour of the participants, using the task video transcripts in conjunction with the task videos themselves to identify problem instances. In some cases, the evidence of a problem might be obvious, for example, a participant wiring up an LED incorrectly, or stating “*I need to find out how to wire up this sensor*”, or “*Is this working? I really don’t know*”. Other evidence might be less obvious, for example, a participant looking puzzled on viewing output in the Serial Monitor.

It is also worth noting that participants were not always aware they had a problem. For example, a participant might introduce a bug but be unaware of it until some time later. All problems were coded, not just those of which participants were aware.

Specific rules

If multiple components or statements of the same type were involved, for example, a participant wired up all 3 LEDs incorrectly, or omitted a semi-colon from each of 3 variable declaration statements, these were coded as 3 separate problems.

Once a problem had been noted, subsequent evidence of the same problem observed *immediately* after the evidence of the initial problem was *not* coded as a new problem. However, if any different activity occurred between evidence of a problem and further evidence of it, the second evidence was counted as a new problem, as it proved too difficult to track problems as *unique* problems (see next section). Often the problem might also have changed slightly, for

example, a participant may have formed a hypothesis that affected their assessment of the current situation and their challenge in addressing it.

Difficulty in tracking problems

I originally intended to track problems as *unique* problems, however, in practice this proved impractical. Problems were frequently nested. A single problem might subsequently decompose into further sub-problems, or lead to new, different problems before the original problem had been resolved. For example, when addressing an issue of not knowing how to wire up the sensor, the participant might find a wiring diagram online that showed how to do this, but subsequently have trouble deciding whether the configuration was appropriate for the current prototype, or determining which way around the sensor in the diagram was oriented, in order to map the connection information to the seating of the actual, physical sensor component in the breadboard.

Participants also did not always deal with problems in a linear fashion. Sometimes a participant would leave a problem to deal with something else, possibly coming back to it later, but possibly not. For example, a participant might not be able to interpret the output in the Serial Monitor, resulting from incorrect wiring of the sensor, but decide to move onto adding and programming the LEDs regardless, and then later decide to return to looking at the sensor output.

3.2.6.3 Problem types

A scheme comprising 3 codes was developed to categorise problem instances by *problem type*. Different, more complex categorisation was considered, however, these three categories felt useful enough to proceed with, as descriptors of the most fundamental characteristics of impediments to participants' progress.

- **Obstacles** were coded on evidence of *barriers to overcome*. These were often due to knowledge gaps.
- **Breakdowns** were coded on *error in action or thinking*, that is, when the participant said or did something that was incorrect.
- **Bugs** were coded for *faults introduced*, most often in the circuit or the program.

The coding scheme evolved from notes made when applying the problem code, but was heavily inspired by Ko and colleagues' work on software error classification (Ko and Myers 2005) and *learning barriers* (Ko, Myers, and Aung 2004). I was familiar with the concept of *learning barriers* as a schema for classifying hurdles encountered by learners when programming, and the potential for *breakdowns* to occur when they are attempting to overcome them. As I chose to include the concept of a barrier within my coding scheme, but not to use the learning barrier classifications (see section 2.2.2), the term *obstacle* was chosen instead, to avoid confusion. Table 3 shows each code and its definition, as well of examples of situations where each code would be applied.

Table 3. Problem Type coding scheme

Code	Definition	Examples of code application
Obstacle	Barrier to overcome. Often due to inadequate knowledge	- Not knowing how to connect the sensor. - Not knowing how to declare a variable. - Not knowing what readings in the IDE's Serial Monitor mean. - Not knowing how to diagnose the cause of unexpected failure or erroneous output
Breakdown	Error in action or thought	- Miswiring the sensor. - Not adding a semi-colon at the end of a variable declaration. - Wrong diagnosis of bug symptoms.
Bug	Fault introduced, usually in the circuit or program. Usually the result of a Breakdown.	- Sensor connected to a digital pin, instead of an analog pin. - Syntax error in variable declaration, e.g., missing semi-colon

Chains of problems

As has been shown in previous work (Ko and Myers 2005; Kulesza et al. 2009) there can be *chains* of problems, for example:

- *An obstacle might lead to a breakdown:* in trying to overcome an obstacle, a user may make a wrong decision, reach a wrong conclusion, or perform a wrong action.
- *An obstacle might lead to another obstacle:* in trying to overcome an obstacle, a user may encounter another one.
- *A breakdown might result in another breakdown:* wrong thinking—for example, an incorrect hypothesis—can lead to wrong action.
- *A breakdown might result in a bug:* wrong action might introduce a fault.
- *A breakdown might result in an obstacle:* wrong action or thinking can cause another barrier for the user to overcome.

Table 4 illustrates how these chains of problems might occur:

However, a breakdown might not necessarily result in a bug or a further obstacle. A participant might state something incorrect, for example, a misdiagnosis of the cause of unexpected prototype behaviour, but then not act upon it; a participant might incorrectly declare an unnecessary variable but then not actually use it.

Table 4. Example chain of problems (partial), showing Problem Type codes

	Activity / evidence	Problem Type
1	A participant does not know how to wire up the sensor.	Obstacle
2	They find a circuit diagram online that shows them how to do it but despite this information, they miswire the sensor, swapping two of the connections	Breakdown
3	This error results in circuit problems (faults) that they will need to fix although they are not yet aware that there is anything wrong.	Bug
4	On viewing the sensor reading values in the Serial Monitor, they note that the values are unpredictable.	Obstacle
5	The unpredictability of the readings is due to the incorrect sensor connections, but the participant wrongly concludes that it may be because they did not use a resistor with the sensor.	Breakdown
6	They subsequently—and wrongly—connect a resistor to the sensor, between the Ground leg and the analog pin.	Breakdown
7	This error results in yet another problem (fault) they will need to solve if they are to succeed in the task (although, they still show no evidence of realising they have done something wrong).	Bug
8	On viewing the new sensor reading values, they note that the values are different, yet still unpredictable. They do not understand why.	Obstacle

3.2.6.4 Problem locations

A *Location* coding scheme was applied to the problems dataset, specifically as sub-codes of problem type codes, to record *where* participants experienced problems. This coding scheme, also inspired by Ko & Myers' work analysing the cause of programming errors (Ko and Myers 2005), eventually comprised four location codes. Although I began with two codes—*Circuit* and *Program*, the scheme evolved inductively during the process of coding, with two further codes added—*IDE* and *Circuit+Program*—as a result of encountering problems within the dataset that could not simply be coded with either of the two existing codes.

Table 5 lists and defines these codes, and provides examples of the circumstances under which each code might be applied.

Where participants showed evidence of missing or incomplete knowledge, the location codes were assigned according to the area of knowledge. For example, if a participant searched for an example program, the 'Program' location code was assigned. If the participant was unable to figure out where to find a particular function/option in the IDE, the 'IDE' location code was assigned. A problem could be coded with more than one location, if applicable.

Table 5. Problem Location coding scheme, with code abbreviation in brackets beneath the code name

Code	Definition	Examples of code application
Circuit (C)	Obstacle, breakdown or bug involves understanding or manipulation of the circuit	- Not knowing how to connect the sensor (Circuit obstacle). - Miswiring an LED (Circuit breakdown; Circuit bug).
Program (P)	Obstacle, breakdown or bug involves understanding or manipulation of the program	- Not knowing how to declare a variable (Program obstacle). - Introducing a syntax error in an If statement (Program breakdown; Program bug).
IDE (I)	Obstacle, breakdown or bug involves the use or function of the IDE	- Not knowing where to find the Upload option in the IDE (IDE obstacle) - Clicking on the wrong toolbar button in the IDE (IDE breakdown)
Circuit+Program (B)	Obstacle, breakdown or bug involves manipulating, understanding or interpreting interaction between both program and circuit	- Not being sure why LEDs turn on or off at unexpected times (Circuit+Program obstacle) - Misdiagnosing bug symptoms in the Serial Monitor (Circuit+Program breakdown)

Table 6 illustrates how these codes might be applied to a chain of problems. Figure 17 shows a coded portion of a transcript.

Table 6. Example of Problem Location codes applied in conjunction with Problem Type codes

	Activity / evidence	Location & Problem Type
1	A participant does not seat the Ground leg of the sensor properly in the breadboard.	Circuit bug
2	When they view the sensor readings in the Serial Monitor, the readings sometimes drop to zero. They have trouble deciding whether this shows evidence of a problem, or is to be expected.	Circuit+Program Obstacle
3	They wonder if they might have done something wrong in the program, causing this behaviour.	Circuit+Program breakdown
4	They subsequently act upon the incorrect hypothesis by modifying the program.	Program breakdown
5	In modifying the program, they introduce a syntax error.	Program bug

Activity	Obstacle	Breakdown	Bug
Stares at the prototype, thinking	B		
Pushes the sensor, watching the LEDs.			
Goes back to the datasheet, and scrolls through it.	C		
"I'm looking in the datasheet again, to see if there's any indication of, uh, what the application circuit is, without getting into too much complicated stuff. Which, I've come back to the same page again, and (shakes his head) I'm not seeing anything new."			
Looks at an application circuit diagram on the datasheet			
"I can see there is a diagram with this particular sensor, but there's a voltage divider, and I don't really understand why there's a voltage divider"	C		
"You know what? I'm just going to randomly put in an extra resistor, and make a voltage divider, because the datasheet says there is one, so maybe that will help, I don't know"		C	
Takes a resistor and starts to change the wiring, adding in the resistor to the sensor		C	C
"So I've got one resistor going to Ground, which should pull it down, does pull it down, and one resistor coming out of here"		C	

Figure 17. Portion of transcript coded with problem type (Obstacle, Breakdown, Bug) and location (Code abbreviations: C=Circuit; B=Both, i.e., Circuit+Program)

3.2.6.5 Problems overcome

Problems were analysed for whether participants managed to overcome them. A problem was counted as overcome if it was resolved. For this analysis, only obstacles and bugs were considered. As breakdowns were errors in action or thought, they could not be resolved—only the obstacles that led to them, or the bugs that resulted from them, could be resolved.

Obstacles were counted as overcome if the participant managed to find a solution to the problem, whether by finding information to fill their knowledge gap or reasoning until correct understanding was reached. For example, an obstacle of a participant not knowing how to connect the sensor was counted as resolved if the participant managed to find out how to do this. If they subsequently made a mistake when applying the information that they had found, the obstacle was still counted as resolved, as the knowledge gap had been filled, irrespective of whether the participant had applied the knowledge correctly

Bugs were counted as overcome if the participant rectified or removed the fault or transformed it into a new, different bug. For example, a syntax error bug introduced was resolved if the participant managed to rectify the syntax. If the participant introduced a different syntax error when rectifying the original bug, the original bug was still counted as overcome, because in effect, that particular bug no longer existed.

Problems were usually overcome by trial-and-error (trying things out or guesswork), existing knowledge (participants using what they already knew to solve a problem or reason towards it),

or looking things up (using additional resources such as online examples or information, or help content in the IDE to find a solution)

Each problem in the transcript was coded according to whether it was overcome or not. The transcripts and videos were used to trace whether each problem was overcome.

3.2.6.6 Task success

Task data were analysed for whether each participant had completed the task *successfully*, i.e. achieved the goal of the given task brief—*task success* was defined as the participant having successfully constructed a working prototype that met the specification in the task instruction sheet/brief, and with the circuit or program containing no fault(s) that would prevent it from behaving as specified—i.e. both *correct* (see section 3.2.6.7). Participants had demonstrated their prototypes at the end of the task, but the task video recordings were also used to determine whether the prototype behaved as specified. The saved programs and the circuit photographs and Fritzing images were also scrutinised for errors that were not evident in the prototype’s runtime behaviour—in some instances, a prototype might appear to behave as specified, but the program might contain an error that was not obvious at runtime, for example, the use of an incorrect operator in the conditional statements (> instead of >=) might leave a gap that was not perceptible to the eye, due to the speed at which the program statements were executed.

3.2.6.7 Correctness of circuit and program

Participants’ programs and circuits—using the circuit photographs and Fritzing diagrams—were analysed for *correctness*. A circuit or program was considered to be correct if it contained all of the elements required in order to meet the brief, configured in such a way as to meet the brief, and contained no bugs (faults) that would prevent the prototype from behaving as specified, irrespective of whether this failure was visible to the eye at runtime. Correctness did not equate perfection—extraneous elements that did not affect the prototype’s runtime behaviour, for example, program variables that had been declared but not used, were not counted as faults in this analysis, as they had no effect on behaviour.

3.2.6.8 Cause of task failure

Task data were analysed for the *cause of task failure*. The cause of task failure was considered to be the *location* (circuit, program, IDE) of the first bug (fault) introduced that the participant failed to resolve which would prevent the prototype from working as specified, regardless of any subsequent, co-existing bugs still unresolved at the end of the task. Note that a participant need not have been aware of a bug's existence, in order for it to be taken as the cause of failure. The cause of failure was determined through analysis of the task transcripts.

3.3 Results

I will now describe the results of the study, in roughly the order of analysis outlined in the previous section.

3.3.1 How many problems? (RQ1)

All participants experienced problems, some more than others, showing that each participant's progress was impeded in some way (Figure 18).

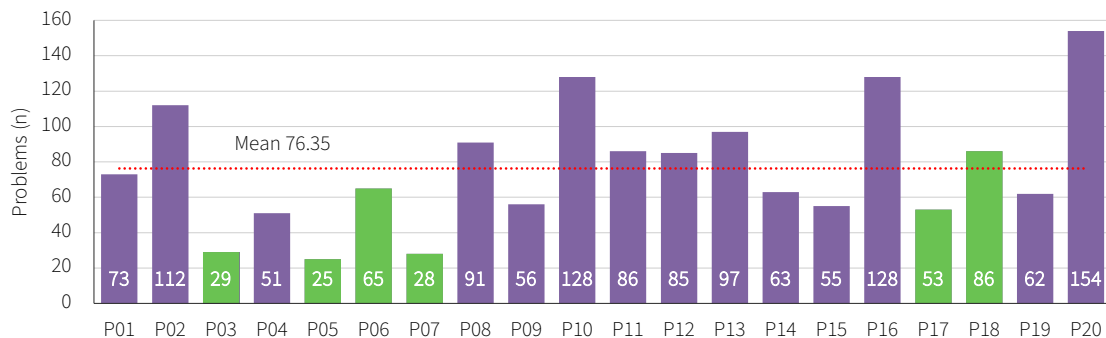


Figure 18. Total number of problems per participant. Participants whose columns are green *successfully* completed the task, i.e., developed a working prototype that met the task brief/specification.

3.3.2 What types of problems? (RQ1)

All participants experienced obstacles, all but one experienced breakdowns, and all but two introduced bugs—most participants (18) experienced all three types of problems. Participants encountered a mean of 41.60 obstacles (SD=14.17), 21.05 breakdowns (SD=13.4), and created

13.7 bugs (SD=9.85) over the 45 minutes they worked on the task. This means that participants struggled a great deal, even though the task was based on a relatively simple Arduino project aimed at learners.

I then investigated whether task success was linked with how many problems were encountered (Figure 19).

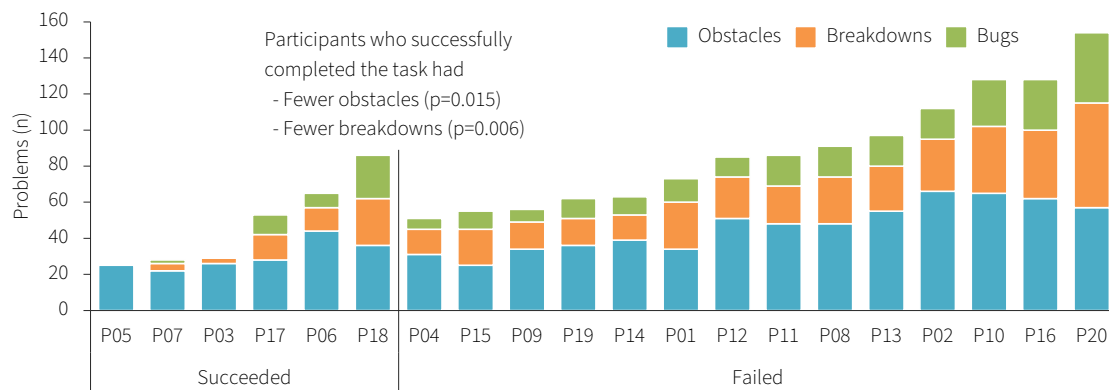


Figure 19. Number of each problem type, per participant, grouped by task success, ordered by total number of problems (obstacles + breakdowns + bugs)

Only six of the twenty participants achieved task success. A Mann-Whitney test showed that the six participants who succeeded had significantly lower total numbers of obstacles ($U=13.00$, $p=0.015$) and breakdowns ($U=10.00$, $p=0.006$) than the fourteen participants who did not succeed. Furthermore, although not significant ($U=18.00$, $p=0.051$), successful participants also marginally introduced fewer bugs. It appears that the successful participants were simply better in some way at physical computing development—either knowing more, or doing fewer things wrong—than their unsuccessful counterparts.

3.3.3 Where did problems occur? (RQ1)

I was interested in where participants' problems were located. Figure 20 shows the count of problems in each location. Most problems related to programming, then circuit construction, then 'circuit+program', with only a few related to the IDE.

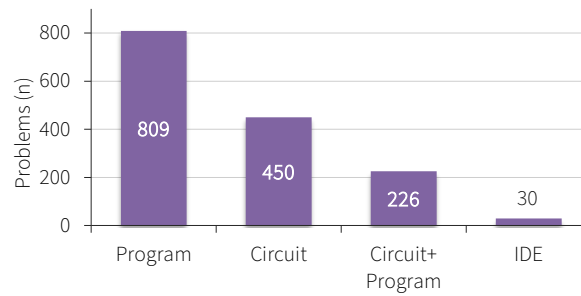


Figure 20. Number of problems by location

Breaking these locations down further, by problem type, reveals that the overwhelming majority of obstacles (49%) occurred in relation to the program (mean=20.40, SD=8.93), followed by 28% associated with circuit construction (mean=11.55, SD=6.36), while 20% of obstacles occurred in the interaction between the program and circuit (mean=8.25, SD=7.87). The same pattern also held true for breakdowns: 52% occurred in the program (mean=10.95, SD=8.41), while 31% of breakdowns were circuit-related (mean=6.45, SD=5.97). This means that participants carried out more wrong actions, and made more incorrect assessments and factually incorrect statements, when they were programming, than when they were constructing the circuit. I also found that bugs introduced by participants related overwhelmingly to their program (66%) instead of their circuit (33%).

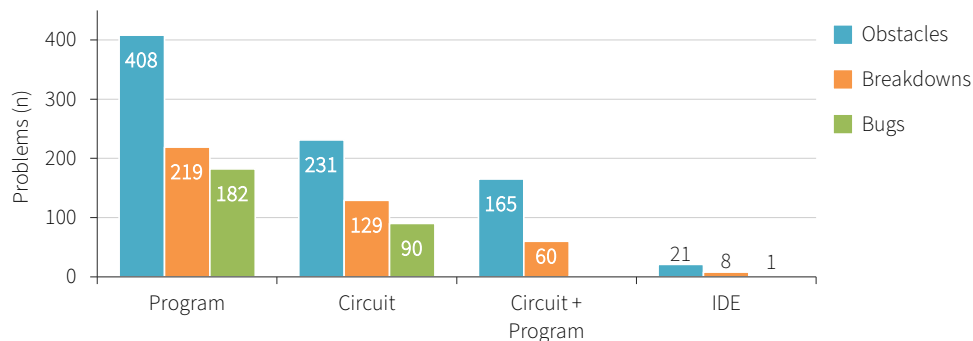


Figure 21. Problem types by location

Figure 21 shows the distribution of obstacles, breakdowns and bugs in the circuit, program, circuit+program and IDE. Only very few obstacles (3%) stemmed from use of the IDE (mean=1.05, SD=1.39). This echoes findings from end-user programming research which showed that users tend to have few information gaps about features of the programming environment and that the majority of problems arise due to issues in problem solving on a strategic level, that is, knowing how to test or debug, or what to do next (Kissinger et al. 2006).

3.3.4 Did self-rated expertise and self-efficacy have an effect? (RQ2)

Recall that I was interested in whether background factors affected the number, type and location of the problems that participants experienced, and whether they were able to succeed in the task. As mentioned, only six participants successfully built a working prototype that met the specification, while fourteen failed to do so.

To answer this research question (RQ2), the problems analysis results were imported into SPSS, along with the data from the background questionnaire and self-efficacy questionnaire. I then looked for relationships between participants' task performance—firstly in respect to task success and then the type and location of problems encountered—and their self-ratings of expertise and self-efficacy.

Using participants self-ratings of their own expertise and self-efficacy for this analysis enabled me to see whether participants' perceptions of their own abilities had any correlation with their actual performance. While not an objective measure, the self-ratings of expertise do provide some indication of how *skilled* participants felt themselves to be; administering tests of competence in programming, electronics and physical computing would not have been practical within the constraints of the study. Participants' self-efficacy ratings are an indication of how *confident* they are at being able to *apply* their skills—expertise—to a particular end, which in this case was specified as an Arduino development task of moderate complexity involving a temperature sensor and three LEDs. The greater a person's self-efficacy in terms of a particular task, goal or activity, the more they believe that they will be successful.

Figure 22 shows participants' individual self-efficacy scores, and whether or not they successfully completed the task (indicated in green), while Figure 23 shows their self-ratings of expertise, and Figure 24 the number of problems each experienced.

I found no significant relationships between task success and self-efficacy, nor between task success and self-rated expertise in programming, electronics, or physical computing.

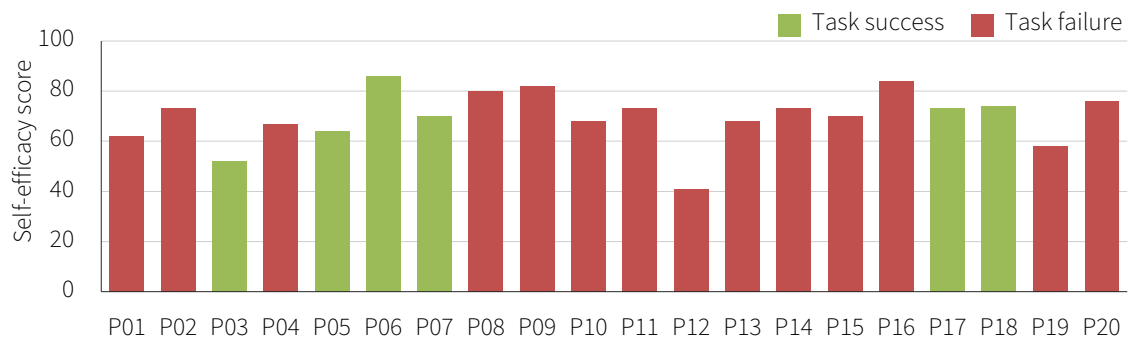


Figure 22. Participants' self-efficacy scores (out of 100), and task success/failure

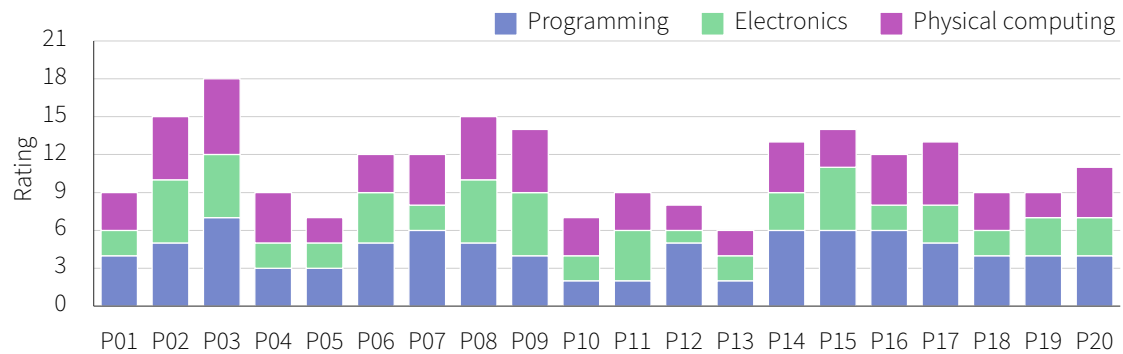


Figure 23. Participants' stacked self-ratings of expertise in programming, electronics, and physical computing (each out of 7)

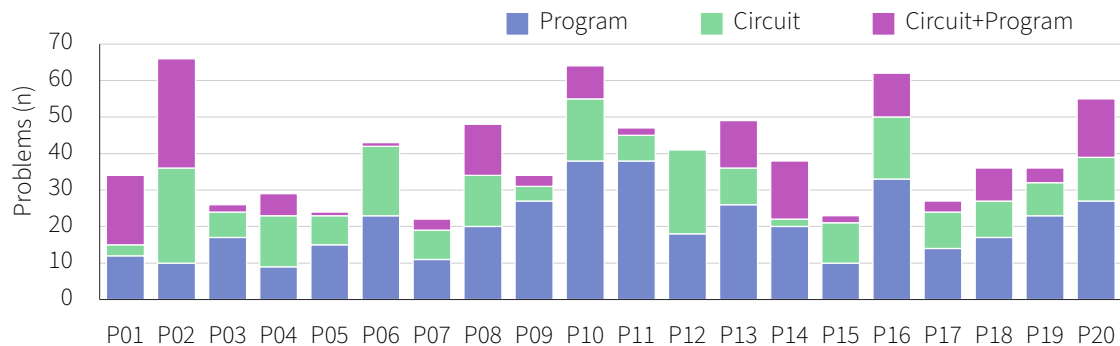


Figure 24. Stacked count of problems encountered by each participant, in the program, circuit and circuit+program locations

Considering that, on average, participants rated their programming expertise higher than their electronics expertise, I was surprised that they experienced more program-related than circuit-related problems. I did not find any significant correlation between their electronics expertise and how many circuit-related obstacles, breakdowns or bugs they had in constructing the prototype, or a relationship between their self-rated programming expertise and their program bugs. Although not significant, there was a marginal relationship between programming expertise and program-related obstacles ($r=-0.431$, $p=0.058$) and breakdowns ($r=-0.400$, $p=0.081$).

Taken together, this means that, in general, participants in this study were poor judges of how good they are at constructing physical computing prototypes.

3.3.5 Were problems overcome? (RQ3)

It might be tempting to deduce that programming was the major challenge for participants in the task, given that most of the problems were programming related. However, the number of problems encountered and where they occurred does not show the *severity* of problems or whether they were successfully resolved (overcome). Some problems might be more easily overcome than others by end-user developers. I now turn to my analysis of whether these problems could be overcome.

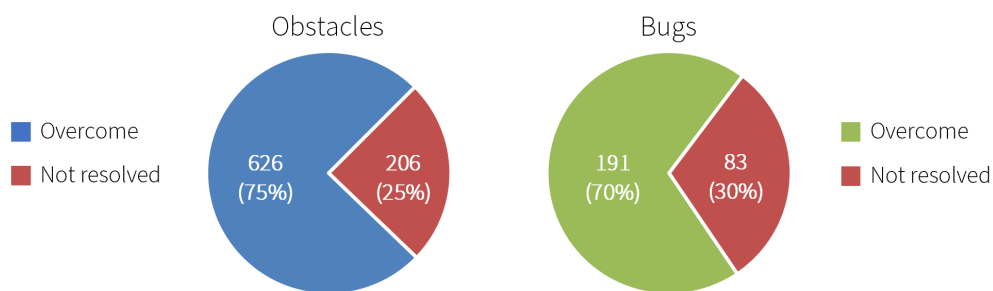


Figure 25. Number and proportion of problems (obstacles and bugs) overcome, or not resolved.

For this analysis, I looked only at obstacles and bugs, since they represent faults which can be overcome, whereas breakdowns manifest as actions or spoken thoughts that cannot be 'undone'. Initially, it appeared that a large number of all obstacles and bugs were overcome by participants, wherever their location (Figure 25)—it is worth noting also that some bugs were also marked as resolved when participants transformed them into new bugs. However, when obstacles involved the interaction of the circuit with the program, less than half of these were resolved (46%), highlighting that these types of problems seemed to be particularly challenging.

I then investigated differences between participants who successfully completed the task and those who were unsuccessful (Figure 26). Successful participants overcame 97% of their obstacles and all of their bugs. Unsuccessful participants, on the other hand, only overcame 68% of their obstacles and bugs (69% and 64% respectively).



Figure 26. Percentage of problems (bugs + obstacles) overcome, by each task success group

3.3.5.1 Cause of task failure

I looked at the types of bugs that caused fourteen participants not to complete the task (Table 7). Recall that the cause of task failure was taken as the first bug that participants failed to resolve that would prevent the prototype from working as specified. I made one exception to this rule—P13 had an otherwise perfectly constructed circuit and program, but because they could not figure out how to view the readings from the sensor in the IDE, they could not establish what temperature thresholds to set—an IDE-related obstacle they could not overcome, which ultimately caused their task failure.

By far the main cause of failure was fault in circuit construction—ten participants—although in nine of these cases, the program was also wrong or incomplete.

Table 7. Participants' task performance and success

Ptc	Task success	Location of first unresolved problem	First problem that was not resolved that would cause task failure	Circuit correct & complete	Program correct & complete
P01	N	Circuit	Sensor signal & Ground swapped	N	N
P02	N	Circuit	No resistors with LEDs	N	N
P03	Y	-	-	Y	Y
P04	N	Circuit	Mis-seated sensor (Ground leg not in breadboard)	N	Y
P05	Y	-	-	Y	Y
P06	Y	-	-	Y	Y
P07	Y	-	-	Y	Y
P08	N	Circuit	No resistors with LEDs	N	N
P09	N	Program	Mixed up variables in adapting copied code	Y	N
P10	N	Circuit	LEDs wired to Power & digital pins, no ground	N	N
P11	N	Circuit	No resistors with LEDs	N	N
P12	N	Circuit	LED connected to sensor	N	N
P13	N	IDE	Did not know how to display serial output	Y	N
P14	N	Circuit	No resistors with LEDs	N	N
P15	N	Circuit	No resistors with LEDs	N	N
P16	N	Program	Wrong operator in conditional statement ().	Y	N
P17	Y	-	-	Y	Y
P18	Y	-	-	Y	Y
P19	N	Program	Wrong operator in conditional statement (> not >=)	Y	N
P20	N	Circuit	No resistors with LEDs	N	N

Four participants failed the task due to at least one unresolved fault in their program code, however, all four did manage to construct the circuit correctly. These participants did much better than the rest of the unsuccessful participants, in both overcoming obstacles and resolving bugs, not just those involving the circuit, but particularly those obstacles involving interaction between the circuit and the program: participants who correctly constructed the electronic circuit overcame 63% of 'circuit+program' obstacles, while participants who did they overcame only 35% (Figure 27). This suggests that circuit construction is something that end-user developers could really benefit from support with.

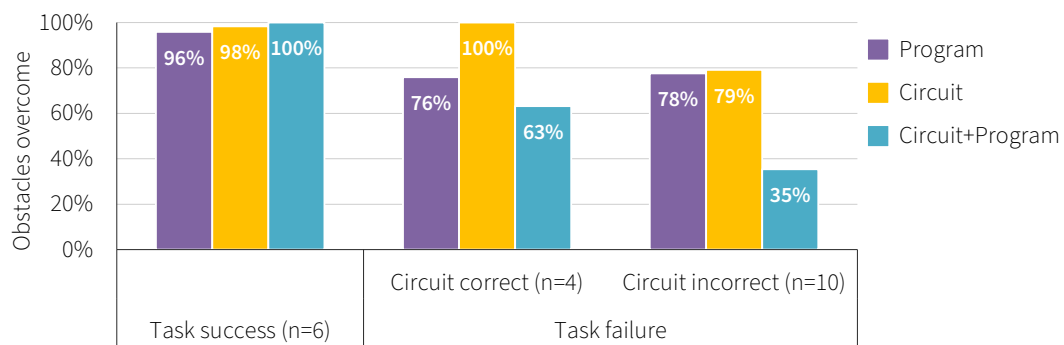


Figure 27. Circuit correctness as a factor in performance in overcoming obstacles

I wondered what activities caused the fatal problems in developing these physical computing prototypes. I present the analysis in the next section.

3.3.6 What went fatally wrong? (RQ3)

I now present a detailed analysis of what participants did which caused them to not complete the task, that is, problems that led eventually to task failure or were very difficult to address. It should be noted that while the primary cause of task failure was taken from a single problem, often a chain or series of bugs were introduced, each further compounding the difficulty of overcoming them.

Program construction

Four participants constructed the circuit correctly but had some fault(s) in their program that prevented them from successfully completing the task. Common faults included using the wrong temperature thresholds in conditional statements, incorrect conditional logic, and numerous problems with variable declarations, assignment and referencing.

For example, participant P16 forgot to add a statement to read the sensor in their program and then referred to the wrong variable in their conditional statements. As a result, the participant saw temperature readings that always remained at zero, regardless of whether they touched the sensor. Attempting to remedy this, they copied code from an example, but the code that they copied did not address the previous two bugs. To compound the issue, they also forgot to change the variable names in the code that they had copied, so now these did not match the ones they were already using in their program.

Challenges in learning to program have been explored extensively within the literature and it seems that many participants in this study struggled with very basic and common programming activities too.

Circuit construction

The most common fatal error that caused ten participants—half of the sample—not to succeed in the task, was some kind of fault in circuit construction. I looked in more detail at what went wrong in these cases.

A high number of breakdowns involved *miswiring* i.e., incorrectly connecting circuit components to the circuit. I observed 87 of these miswiring breakdowns. All but one of the unsuccessful participants encountered these mistakes, and for five participants this caused bugs which prevented them from completing the task successfully. The most common miswiring breakdown was connecting the legs of the temperature sensor or LEDs to the wrong types of Arduino pin. For example, P01 accidentally miswired the sensor very early in the task, resulting in unpredictable sensor readings. Unsure of whether these readings were “normal”, and wondering if there might be an “accuracy” problem, they searched online for ways to programmatically make the readings more reliable, and copied in unnecessary code, to no avail. Forum posts found online—none of which were relevant to the bug—led this participant to make yet more changes to both their circuit and program, none of which addressed the original miswiring bug, and eventually they gave up:

"It's the world. It's just unpredictable in the world. [...] It's technically doing what I want it to do, but it's the world that's breaking, as in, I can't get it to get to the right temperature" (P01).

A particular type of miswiring—*poor seating* of a component or wire into the breadboard—was observed for three participants. In one case, the participant did not realise that a poorly seated

sensor—connection had not been properly established with the contacts inside the breadboard and therefore the rest of the circuit—was the cause of the unpredictable sensor readings they observed in the IDE's Serial Monitor, which intermittently dropped to zero.

"So why does the sensor don't work? [sic] It should be work. [sic] So it goes to zero. I didn't change anything with the sensor" (P04).

Like P01 they were unsure as to whether this behaviour was normal, so looked for help online, however, incorrectly assuming that an error in the program was the cause, searched for program-related rather than circuit-related guidance. They made several trial-and-error changes to their program—none of which addressed the bug—to see if the readings would improve. They did not, and this bug went unresolved, eventually leading to task failure:

Another kind of circuit construction error that prevented task success involved five participants not using resistors with the LEDs. In this task, the missing resistors caused a very insidious problem, by affecting the behaviour of the temperature sensor, making readings very unpredictable—the large amount of current drawn by the LEDs affected the sensor readings: values rose higher and quicker than normal, and dropped more slowly.

"I mean, it should work. The problem is just that the sensor doesn't seem to be very responsive. Because it starts at 150 and when you put your hand there it went over 180, and never came back to 150" (P20).

None of these five participants ever localised or fixed this bug. Instead, unable to determine the fault location in the circuit, three tried to address the problem through extraneous program code, or modifying otherwise perfect code, while one also added extraneous resistors to the temperature sensor.

I also noticed that four participants chose too high a value of resistor to use with the LEDs. Although this did not prove fatal to the success of three of these participants—the LEDs lit up but were dimmer than they should have been—one participant wired a single resistor of such a high value to all of their LEDs that two did not light up and the third only blinked intermittently. To address this, they disconnected the resistor from two of the LEDs, causing the same insidious sensor readings problem mentioned previously—a problem they never resolved.

Two participants connected the LED cathodes to digital pins and anodes to 5V power (rather than anodes to digital pins and cathodes to ground). If these connections were reversed, an LED would not work at all, however, with this bug, the LEDs will work in an unexpected way: the LED

lights up when it should be off and turns off when a signal is sent to turn it on. Both participants assumed that the error lay in their program and tried to address it there, making and testing several changes to the temperature thresholds in the conditional statements that specified the logic controlling the LED behaviour:

"I'm sure my code is not right, about the If statements. I'm pretty sure the If statements are not right. But I'm pretty sure that if I change 24 to something else..." (P10)

While P10 was correct in their hypothesis that there was something wrong with their `If` statement, as it did indeed contain a bug, this was not at the root of the erroneous behaviour they were seeing. Neither participant resolved this circuit bug, and both failed to complete the task.

Testing

Testing a physical computing artefact can be more complex than testing a program alone. In two instances, participants who had constructed their prototype correctly, touched their temperature sensor and the LEDs did not light up. In fact, they had cold fingers, that is, their test 'input' was bad. In one instance, this led a participant to believe there was a fault when there was not. In software, a more appropriate test strategy would be to use a variety of test inputs including edge cases.

Debugging

Professional software development environments usually provide a debugger, which helps programmers to locate and fix faults, and end-user programming environments have started to do similar. Unfortunately, the Arduino platform does not yet have analogous debugging tools and thus it was sometimes difficult for participants in this study to identify what the problem was.

However, characteristics of some problems proved helpful in guiding participants towards the source of a fault. One particular miswiring fault that four participants were able to identify and fix was when they erroneously reversed the power and ground connections of the temperature sensor. This error led to the component heating up, and as a result they felt momentary discomfort when they touched it: although very unexpected, this feedback, experienced in the location of the fault, helped them to localise the fault immediately to that particular component in the circuit.

A pattern of misdiagnosis leading to bugs: I have already highlighted the insidious problem resulting from missing resistors. The only way that participants were able to spot this problem was by noticing that the sensor readings were incorrect when viewing them in Arduino IDE. However, perhaps because their focus was on the programming environment at this point, they usually tried to debug this issue by making changes to their program code. This proved to be a fairly common pattern: bug → misdiagnosis (breakdown) → wrong action to resolve (breakdown) → bug. A similar pattern has been observed in studies of end-user programming, although here it is also more complicated in that a wrong hypothesis as to the cause of failure can lead to the end-user developer making changes in a completely different location altogether.

Summary

Why did it go so wrong for many of the participants? The study showed that problems in physical computing are to be expected, even for users who are eventually successful, but it also showed that some circuit construction errors were particularly hard to identify and remedy. Five participants did not even realize that a circuit-related error was preventing their prototype from working and attempted to fix the perceived fault by changing their program code. Obviously, that proved in vain and in fact caused four of them to introduce more bugs into their program. This might also explain why I observed so many program-related obstacles, breakdown and bugs, and the high proportion of problems that were associated with the interaction of circuit and program—once participants started to incorrectly believe that the issue was in the program instead of the circuit, they often created further problems in this location. A major contributory factor here might be that testing and debugging physical computing prototypes are both very challenging and appropriate support tools are not currently available.

3.4 Discussion

The study revealed the following:

- All participants encountered problems when developing the device—all experienced obstacles (barriers), and the vast majority also experienced breakdowns (errors in fault or action) and introduced bugs (faults), in their circuit, program or both (RQ1).

- While most problems occurred in programming the device (RQ1) the majority of task failures—inability to develop a working prototype that met the task brief—were primarily due to circuit-related problems (RQ3).
- Circuit-related task failures were mainly attributed to two types of bugs: *miswiring*, for example, providing the wrong connections from the Arduino board to the sensor, and *missing components*, for example, failing to use resistors with the LEDs. Participants had serious difficulties localising the circuit faults that caused them to fail the task. (RQ3).
- In diagnosing the symptoms of these bugs, participants did not always realize that there was a fault or error in their circuit and often incorrectly tried to fix the perceived problem by modifying their program, leading to new program bugs. In some cases, they also chose to modify a different part of the circuit, also introducing new circuit bugs. (RQ3),
- Background factors such as self-efficacy and self-rated expertise did not predict whether participants would complete the task, nor the number, type and location of problems they experienced (RQ2).

A number of limitations should be acknowledged for this study—these are detailed in section 7.2, and include, for example, the small sample size. However, there appear to be clear opportunities to help end-user developers overcome their problems, drawing on insights from studies of end-user developers’ problems in software development. Targeting support at the most severe problems—the circuit bugs that led to so many new problems and prevented participants from successfully completing a working prototype—seems a logical approach, but while the analyses described in this chapter identifies the type and location of problems most likely to cause end-user developers the most trouble, at this point, we still do not know why they failed to resolve them.

My next task was therefore to find out why participants failed to overcome these bugs. What behaviours did end-user developers employ when troubleshooting and why did these fail? Might different tactics have helped these participants to diagnose and resolve their bugs? If so, what are the critical points at which end-user developers could be caught and guided towards information that might help them troubleshoot?

The next chapter describes work undertaken to answer these questions, namely a deeper analysis of the same data, this time focusing specifically on participants’ *troubleshooting* of problems resulting from circuit bugs introduced during development.

Chapter 4

How end-user developers troubleshoot circuit bugs (Study 1B)

4.1 Introduction

A second analysis of data collected in the previous study, this new study now sought to address the lack of knowledge about *how* end-user developers troubleshoot their physical computing problems and whether their approaches are effective, and in doing so, answer the second research question guiding this thesis: while Study 1A aimed to identify the *problems* that end-user developers encountered when developing a physical computing device (TRQ1), the next analyses of these data—Study 1B—sought to identify and understand end-user developers’ *troubleshooting behaviours* (TRQ2) when attempting to find and fix circuit bugs—the type of bug associated with most task failures (section 3.3.5.1).

In the previous analyses I observed many instances of participants failing to localise their circuit bugs successfully, and introducing numerous new problems in the process of troubleshooting. I now wanted to understand where participants went wrong in their troubleshooting attempts, and why, for half of the sample, inability to resolve this type of bug resulted in task failure. I was also curious to find out what had enabled *some* participants to successfully resolve at least some—if not all—of their circuit bugs.

The decision to now focus on troubleshooting behaviours was informed by the novice and end-user programming literature. Research into end-user programmers’ debugging behaviours (e.g., Kissinger et al. 2006; Grigoreanu, Burnett, and Robertson 2010; Kulesza et al. 2009) has proven valuable in determining avenues of support for them (e.g., Cao et al. 2015), in part by identifying problematic areas to address. For example, work has shown that end-user programmers can

employ unproductive or destructive strategies, or simply do not know which strategies might help them (see section 2.2.3).

Having observed several participants struggle to diagnose their circuit bug-related problems, and having noticed some specific patterns of misdiagnosis, I wondered whether looking deeper at the troubleshooting behaviours of *these* participants might yield insights into how to support end-user developers dealing with these types of problem.

Therefore, rather than collecting new data, this new piece of work built upon the first set of findings with further, deeper analysis of the same data, identifying the behaviours employed by end-user developers when troubleshooting the effects of circuit bugs that they themselves had introduced, and determining whether these are effective.

The study addresses the following thesis-level research question:

TRQ2: How do end-user developers troubleshoot the most significant problems that arise during development, and from what support might they benefit?

This has been broken down into two more-specific research questions, guiding this analysis:

RQ1: How do end-user developers troubleshoot circuit bugs? What troubleshooting tactics do they use?

RQ2: Are end-user developers' troubleshooting behaviours effective in helping to resolve their circuit bugs?

4.2 Method: Data Analysis

This section begins with an overview of the analysis, including an outline of the data segmentation and of the process of coding. Thereafter I describe the segmentation and coding schemes in detail, including rules for application.

4.2.1 Overview

Analysis now focused on participants' *troubleshooting of circuit bugs*—the problems that had proven most severe in preventing the successful building of a working prototype. A participant

was said to be troubleshooting a circuit bug when they showed evidence of noticing the symptoms of one at runtime—irrespective of whether they realised it was a circuit bug causing the symptoms—and made some attempt to investigate or address it, or if they noticed or suspected—correctly—a potential circuit bug when building their prototype and then took steps to investigate or resolve it.

The dataset consisted of the task data collected for fourteen of the twenty study participants—that is, all those who encountered and troubleshot one or more circuit bug-related problems. Data from the other six participants was not analysed because they either experienced no bugs at all, or only experienced program bugs.

The main source of data in this study was the task transcripts from the previous analysis, again supplemented with the task videos. Recall that the transcript spreadsheets contained a written record of participants’ think aloud comments and actions during the task—one spreadsheet per participant. Previous coding—specifically, problem types and problem locations—now made it easy to identify the fourteen participants who had introduced and dealt with circuit bugs.

4.2.1.1 Data segmentation and coding schemes

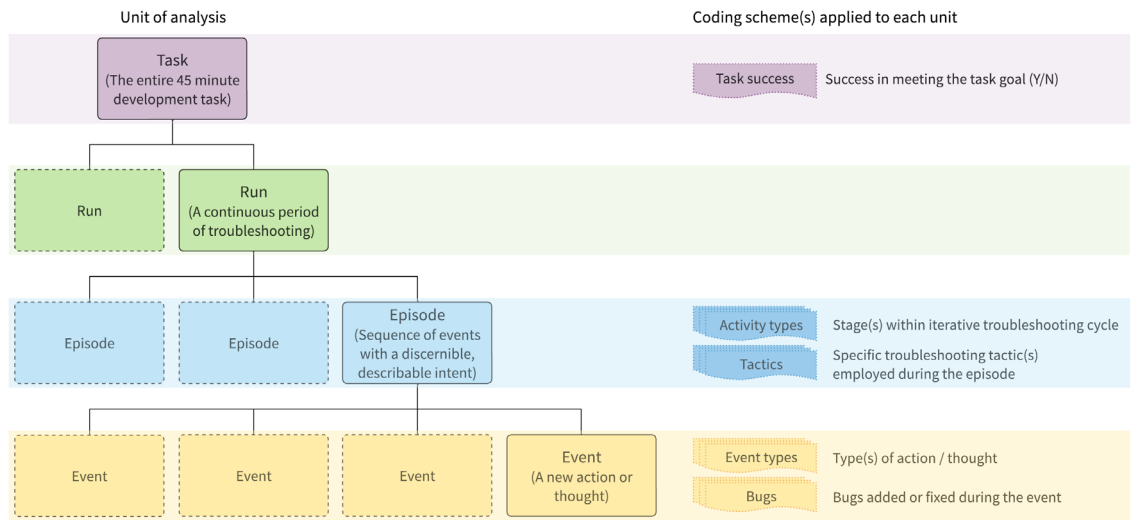


Figure 28. Hierarchy of units of analysis , and the coding schemes applied at each level. A task contains one or more troubleshooting runs. A run consists of one or more episodes. An episode consists of one or more events.

A substantial part of answering the two research questions entailed segmenting portions of the transcripts into specific units, to which a number of coding schemes were applied. Figure 28 shows the organisational hierarchy of segmentation of the task transcript spreadsheet data, and

the coding which was applied at each level. Note that this diagram does not indicate the sequence in which segmentation or coding took place.

- Each participant undertook one *Task*. The coding scheme applied at this level was *Task success*, i.e., whether the participant *succeeded or failed* to complete the task.
- A task contained one or more troubleshooting *Runs* (section 4.2.3)—continuous periods when a participant was observed to be troubleshooting circuit bug-related problems. No coding schemes were applied at this level—runs serve mainly to demarcate periods of troubleshooting.
- A troubleshooting run consisted of one or multiple *Episodes* (section 4.2.4)—sequences of events (see next point) that could be easily described in terms of intentional behaviour. Two coding schemes were applied at this level: *Activity Types* (section 4.2.5), and *Tactics* (section 4.2.6).
- Each episode consisted of one or more *events*—individual comments or action by the participant. The two coding schemes applied at this level were *Event Types* (section 4.2.2), and *Bugs*—new bugs and bugs fixed, coded in the previous analysis (section 3.2.6.3). As with bugs, a *Location* sub-code (e.g., Circuit or Program) was applied to Event Type codes, where appropriate.

4.2.1.2 About the coding process

In the previous study, the application of all coding schemes took place in Excel. In this study, coding began in Excel, but then transitioned to the hand-coding of printed visual representations of the data, before returning to Excel for the remainder of the analysis. As before the approach to coding was highly iterative, involving multiple passes through the dataset, and began with a period of familiarisation, this time focusing attention specifically on evidence of troubleshooting.

The problem-coded transcript spreadsheets, supplemented with the task videos, were used to identify periods within the dataset during which circuit bug-related problems were troubleshoot. Within these portions of the transcripts, *events* (actions or comments) associated with troubleshooting were coded with *Event Type* codes—a coding scheme developed inductively from the data—and refined through iteration and discussion. This coding was then used to identify *episodes* within the transcript spreadsheets. Episodes within a spreadsheet were numbered sequentially, as were runs if more than one was evident.

For coding the episodes with *Activity Types* and *Tactics*, however, rather than working with the transcript spreadsheets, I found it useful to view the data in a different form—the run and episode segmentation and the event type coding were used to create visual representations—*troubleshooting flow diagrams*—of the sequence/flow of troubleshooting episodes. Appendix H shows an example. Beneath the summary of circuit bug(s) that the participant troubleshot, the sequence—and summarised content—of episodes is shown. Each numbered block represents an *episode*, annotated with transcript text to the right. The coloured rectangles within each episode block summarise the *Event Types* that were coded, in the order of their occurrence, for example, a sequence of several *Get help* events in which a participant looked at a number of different web pages online, would be represented by a single *Get Help* block. Event types were dual coded using words and colours in the transcript spreadsheets and the same colours were used in the troubleshooting flow diagrams, for visual clarity, for example, *Change* blocks are always orange, *Inspect* blocks always blue, etc.

Episodes in the troubleshooting flow diagrams were coded with *Activity Type*, and subsequently, *Tactic* codes. The development of the Activity Type codes (section 4.2.5) followed an inductive approach, initially identifying lower level ‘goals’ of activity within the dataset and then later comparing and consolidating them. While coding of Tactics began, deductively, with initial codes and coding rules derived from or informed by the literature, this was supplemented with inductive coding. As in previous coding of the task data, these coding schemes were developed and refined through repeated iteration over the dataset and discussion with my supervisory team until code definitions were stable and rules deemed reliable. Once hand coding was complete, the coding was transferred to Excel, for further analysis.

Although creating the troubleshooting flow diagrams was somewhat laborious, I found this more visual, summarised view very helpful when coding the episodes, given the large amount of data in the transcript spreadsheets. The process of creating these diagrams also provided a further opportunity to review the episode segmentation and amend this where appropriate. The diagrams were also very useful in discussing the segmentation and any specific coding decisions with others, for example, to ensure reliability.

Much of the remainder of the analysis was done in Excel, using formulae and pivot tables, however Microsoft Access was also used to parse and query the spreadsheet data, including to transform it into structures that could be analysed more easily within Excel. My previous experience with Access development made this the easiest way to achieve what I wanted to do.

The segmentation and coding schemes, including the rules of application, will now be described in greater detail.

4.2.2 Event Type codes

Portions of the transcripts in which troubleshooting of circuit bugs took place were segmented into sequential units of analysis, with each unit representing a new action or statement by the participant—each, effectively, an *event*.

A set of troubleshooting *Event Type* codes was developed (Table 8), focusing only on the low-level ‘what’ of participants’ comments and actions, rather than ‘how’ or ‘why’: realising that there is a problem (*fault recognition*), *hypothesising* about something, visually *inspecting* something, making some kind of *change*; *testing the system* in some way, or seeking *help*.

Table 8. Troubleshooting event type codes

Code	Description
Fault Recognition	The participant indicates they know or suspect there is a problem, or they see hard evidence of an error, e.g., a compiler error message, or the sensor feeling hot to the touch.
Hypothesis	The participant states a concrete idea of how to resolve a problem, what may be wrong, or why something is not (or may not be) working. Can include loose hypotheses, e.g., ‘I think I may have done something wrong in the <i>circuit</i> ’.
Get Help	The participant seeks or uses help from other sources. E.g., reading help content or examples built into the IDE, or copying circuit schematics in online tutorials
Inspect	The participant visually inspects some aspect of the circuit, program, or IDE, to evaluate correctness, locate a fault, or understand it. E.g., checking the sensor wiring
Change	The participant makes some kind of change, in an attempt to resolve a problem, isolate the cause of one, or determine whether there is a problem. E.g., swapping an LED with a new one from the parts kit, or changing the sensor wiring.
Test System	The participant tests or evaluates their prototype, or a part thereof, either to determine its correctness, or to isolate the cause of a perceived/suspected problem. E.g., watching LED behaviour, viewing sensor readings printed to the Serial Monitor, or holding and releasing the sensor to see the effect on behaviour or output.

The intention with this coding scheme was to see whether defining some fundamental building blocks of troubleshooting activity would later help in identifying patterns of intentional behaviour within the transcript spreadsheets. With so much data to analyse, this felt like a practical first step. Therefore, rather than using a predefined coding scheme, these codes were developed inductively from the data—familiarity with the task data, including from previous analyses, provided a starting point, but as previously, codes and the rules for applying them were refined over several iterations, and through discussion with my supervisory team.

In applying the codes to a transcript, coding began at the participant's first voiced identification or suspicion of a circuit-related problem (irrespective of whether they *realised* the problem was circuit-related)—and stopped either when troubleshooting ceased, for example, if the problem was resolved or at the end of the task session if resolution did not occur. Multiple Event Type codes could be applied to each unit, and units could be ignored if it was felt that no codes applied. A portion of a coded transcript is shown in Figure 29.

Location sub-codes were also applied, using the same sub-codes used previously (section 3.2.6.4). For example, if a participant hypothesised that a perceived problem was caused by an error in their *circuit*, the *Hypothesis* code was applied to this statement, and sub-coded “C” (for Circuit). When coding *Fault recognition*, a ‘1’ flag was used, rather than a location sub-code.

Only units with some relationship to the troubleshooting were coded. For example, if a participant stopped troubleshooting to attend to another aspect of development—for example, if they wrongly assumed they had solved the problem, or merely decided to move on to something else—then the units in which they were not troubleshooting were not coded. If a participant encountered another problem to investigate/resolve when troubleshooting the original problem, those units *were* coded, being somewhat embedded within the original line of troubleshooting. This meant coding some sequences of units where multiple, co-occurring problems were dealt with at the same time by a participant. In fact, participants sometimes introduced further bugs when troubleshooting, for example through misdiagnosis of the cause of a problem, or slips and mistakes made when implementing a potential fix.

To establish the reliability of the code set, myself and another researcher independently coded a segment of the dataset in which a participant troubleshooted a circuit miswiring bug. In addition to the transcript spreadsheet, we each had the video to refer to for further clarification. Coded transcripts were then compared, and areas of disagreement discussed, before the coding

scheme was adjusted, refining the definitions, and teasing out any ambiguity. Five separate rounds of this took place, involving three researchers and four different segments of the task videos, until agreement of 81.7%, calculated by the Jaccard index, was reached. Given the acceptable level of agreement, I then coded the remainder of the dataset.

Time	Event	Fault recognition	Hypothesis	Get help	Inspect	Change	Test system
00:29:01	"Maybe I made a mistake, because we're already at 29 degrees."	1					
00:29:13	Checks the circuit, specifically the wiring of the LEDs.				C		
00:29:23	Briefly removes and reseats the blue LED.					C	
00:29:27	Holds the sensor briefly, watching the LEDs.						B
00:29:32	Watches the readings in the Serial Monitor.						B
00:29:38	Switches back to the program and starts reading through it, checking it. "I have 11, 12, 13... I'm reading the voltage..."				P		
00:29:44	"It's probably my electronics skills"		C				
00:29:50	Goes back to the circuit and starts checking it. "Let's see..."				C		
00:29:52	Removes the yellow LED.					C	
00:29:55	"Oh, no. It's just remembering which direction I put them in"		C				
00:31:01	Reseats the yellow LED, the opposite way.					C	
00:31:03	Holds the sensor. The blue LED comes on, then starts to flicker on and off.						B
00:30:11	Releases the sensor and stares at the LEDs. "Hmmm"						B
00:30:15	Checks the wiring again.				C		
00:30:26	"Let's see, ok, so I have the ground coming from here, then it's this resistor, then this one goes into here, and it goes back to the resistor, fine... then this one goes back, the shorter"				C		
00:30:49	Removes the blue LED briefly and reseats it.					C	
00:31:04	"And this one (checking) the shorter"				C		
00:31:05	Removes the yellow LED					C	
00:31:08	Removes the blue LED					C	
00:31:10	"Let me just try if the LEDs actually work"		C				
00:31:12	Seats the yellow LED in the position vacated by the blue LED.					C	
00:31:14	Holds the sensor and watches the LEDs.						B
00:31:16	"No." (The yellow LED hasn't come on)						
00:31:19	"Is it that I burnt it by using it the wrong way round"		C				
00:31:21	Removes the yellow LED.					C	

Figure 29. Event codes applied to a portion of a transcript spreadsheet. A letter within a coloured (coded) cell denotes the location subcode (*C*=Circuit; *P*=Program; *B*=Both, i.e., Circuit+Program). Note that rather than a location sub-code, the *Fault recognition* code contains a flag ('1') to indicate the point at which the participant became aware of the problem

4.2.3 Runs

The coded transcripts were used to identify the start and end of troubleshooting *runs*—any period of continuous troubleshooting of a circuit bug-related problem. A run ended either with evidence that troubleshooting had stopped, for example, if the participant indicated that they knew (or thought) they had resolved their problem and continued with further construction of the device, or at the end of the task, if resolution did not occur. As some participants had more than one continuous period of troubleshooting, a transcript could contain one or more runs.

No coding schemes were applied to runs. The main use of runs was to demarcate or identify exactly which portions of each transcript spreadsheet were included in the analysis. This was helpful in summarising these data, for example, when calculating how much time participants had spent troubleshooting, and when creating the troubleshooting flow diagrams.

In total, I identified 26 troubleshooting runs, across the 14 participants.

4.2.4 Episodes

Within runs, using the event codes, I then homed in on troubleshooting *episodes*. Episodes incorporated sequences of activity (events) that could be easily described in terms of intentional behaviour. This segmentation was inspired by Newell and Simon, who describe an episode as “a succinctly describable segment of behaviour associated with attaining a goal” (Newell and Simon 1972, 84).

Working iteratively through the coded transcripts, I looked for obvious changes in activity and intention, based on what participants were doing and saying, marking each transition as the potential start of a new episode.

While, in some cases, episodes included only one event, many included multiple. Consecutive events of the same type undertaken with roughly the same discernible intent were counted as belonging to the same episode. For example, if a participant spent a few minutes looking at several different web pages in succession, trying to find information about the correct wiring of LEDs—i.e., several *Get Help* events, all with broadly the same intent—this was counted as one episode, as, per Newell and Simon’s definition, I could reasonably describe this, succinctly, as ‘looking online for information about the correct wiring of LEDs’. If they then made several changes to the wiring of the LEDs, to correct a perceived/suspected error, this was counted as another episode, this time consisting of multiple, consecutive ‘*Change*’ events. Subsequent running of (and runtime interaction with) the prototype to determine whether the fixes had been successful—i.e., one or more *Test System* events with the same intention—was counted as yet another episode. I also checked whether subsequent or preceding units should be included, for example, a participant explaining what they were going to do immediately prior to doing it.

Often a transition from one type of event to another meant a new episode, as the participant was doing something new/different. However, if two types of event (action) could not be

reasonably be separated in their description, they were both included in the same episode, for example, a participant using an online diagram to guide them during the rewiring of a component involved rapid interleaving of Get Help + Change events.

To establish the reliability of my segmentation, I went through several portions of the transcripts with one of my supervisors—a senior researcher and physical computing expert—until we were confident that segmentation was consistent.

The two coding schemes which applied to episodes—*Activity Types* (4.2.5), and *Tactics* (4.2.6)—will now be described.

4.2.5 Activity Type codes

The troubleshooting *activity types* coding scheme attempted to address the ‘why’ of troubleshooting activity, but at a very high level (Table 9). It was inspired by Katz and Anderson’s General Troubleshooting Model (Katz and Anderson 1987), but is even simpler than that model, focusing on the iteration of activity. Each episode was coded with either a single activity code or, in some cases, multiple activity types.

The coding scheme, shown in Table 9, comprises three troubleshooting activity type codes: *Diagnose*, *Fix* and *Evaluate Fix*. These activity types were developed inductively from the data, and once again refined through much discussion and multiple iterations of application, but it is useful to compare them to the Katz and Anderson model which inspired the decision to focus on what can be seen as three primary goals of activity within the iterative cycle.

Table 9. Activity Types coding scheme

Activity Type	Description
Diagnose	The participant knows or suspects that they have a problem and shows evidence of trying to understand it, confirm its existence, or to identify its cause
Fix	The participant attempts to directly resolve a problem through some kind of action, usually making one or more changes to the circuit, program, IDE, or USB connection between the PC and Arduino board.
Evaluate Fix	The participant evaluates the success of an attempted fix, ideally by conducting a test to establish that the problem has been resolved, but alternatively by double-checking their implementation, through inspection, for example, in comparison to an example.

In *Diagnose*, the participant knows or suspects that they have a problem and shows evidence of trying to understand it, confirm its existence, or to identify its cause—which naturally includes fault localisation. This activity type is characterized by the gathering of information and the processing of that information in order to reach a hypothesis or conclusion, for example, the potential cause of a problem, the location of an error causing a problem, or how to resolve a problem. In this definition, *Diagnose* amalgamates *Understand System*, *Test System* and *Locate Error* in the Katz and Anderson model.

In *Fix*, the participant tries to resolve a problem through some kind of action, usually making one or more changes to the physical circuit, program, IDE, or USB connection between the PC and Arduino board. This activity type relates to *Repair Error* in the Katz and Anderson model when a participant has identified the cause or knows how to fix it, however, I also include cases where a participant makes *speculative* changes, that is, without being sure of the cause of a problem or whether the change that they were making would resolve it.

Finally, in *Evaluate Fix*, the participant evaluates the success of an attempted fix, ideally by conducting a test to establish that the problem has been resolved, but alternatively by double-checking their implementation through inspection, for example, by comparing it to an external resource, such as a wiring schematic. This activity type corresponds to *Test System* in the idealized troubleshooting model.

As mentioned, the Activity Type codes were initially applied, by hand, to printouts of the troubleshooting flow diagrams (Appendix H). Later this coding was transferred to Excel.

4.2.6 Tactics codes

I also used the troubleshooting flow diagrams, in conjunction with the videos, to hand code the episodes with a second coding scheme, now addressing *Tactics*. This was inspired by previous work investigating software debugging strategies and troubleshooting hardware, supplemented with additional codes developed through an inductive process (Table 10).

In this work, a tactic is designed as *an observable pattern of troubleshooting behaviour*. There is some inconsistency within the literature regarding the definition of the term *strategy* and I felt ‘tactic’ to be closer to describing the level of some of the behavioural patterns I observed and considered worthy of representing in the coding scheme—strategic thinking was difficult to

deduce from the think aloud data and therefore coding was often based on what I could reliably deduce from participants' actions in conjunction with their comments (section 7.4.2). Focusing on tactics was also inspired by discussion of tactics in the context of electronic testing procedures (Lesgold and Lajoie 1991) and Grigoreanu and colleagues' definition of a tactic, adapted from Bates (1990), as "*the use of one or more moves with the purpose of more quickly and accurately finding or fixing a bug*" (Grigoreanu, Burnett, and Robertson 2009). While some of the patterns I report as tactics may be closer to what Grigoreanu and colleagues would term *stratagems*, and what others *would* refer to as strategies, I deliberately chose to keep codes at one level for this analysis, focusing simply on observable patterns, referred to as *tactics*.

To develop this coding scheme, I first undertook a review of the literature on end-user and novice programmer debugging strategies (e.g. Grigoreanu, Burnett, and Robertson 2009; Murphy et al. 2008), then looked to the literature on hardware troubleshooting (e.g. Steinberg and Gitomer 1996), and problem solving (e.g., Wickelgren 1977), and finally, reviewed a selection of non-academic literature on software debugging and the troubleshooting of systems and circuits (e.g. Agans 2002; Craft 2013; Tomal and Agajanian 2014). From these sources—far more than are cited here—I collated an initial set of codes, which I adapted, where necessary/appropriate, for use in a physical computing context. I refined the code set through iteration across the dataset, looking for relevant patterns in the data, inductively developing further codes where a pattern of troubleshooting behaviour was evident but none of the existing codes applied. Candidate codes were discussed at length with one of my PhD supervisors, until there was consensus in the reliability of the definitions and under what conditions the codes might be applied. The final code set can be seen in Table 10.

I applied the tactics codes to troubleshooting episodes in the troubleshooting flow diagrams, looking for evidence in participants' comments and actions, for example, an episode in which a participant searched online for information about the TMP36 temperature sensor would be coded with *Get Help*. Where appropriate, episodes could be coded with multiple tactics, for example, an episode in which a participant removed an LED from the breadboard and re-seated it the opposite way around, to see (without any certainty) if that would solve the problem, would be coded with both '*Speculative Change*' and '*Reverse orientation*'. Similarly, an episode in which a participant used a diagram found online to guide the wiring of the temperature sensor would be coded with the '*Get Help*' tactic and the '*Copy example*' tactic. Once hand coding of episodes was completed, it was transferred to Excel, for further analysis.

Table 10. Tactics coding scheme and frequency of code application (total count of episodes coded) across the sample

Tactic	Description
Run and analyse	Run and/or interact with the prototype and analyse the output or behaviour
Inspection	Visually inspect the program, circuit or IDE settings for error
Speculative change	Make a change to resolve a problem, without being sure of the cause and/or how to fix it.
Serial Monitor	Use the IDE's built-in Serial Monitor tool to view program output
Get help	Search for, or use, external resources
Causal reasoning from output	Reason backwards from faulty behaviour/output to its possible cause
Isolation	Reduce a part of the system, for testing, or test an isolated part of the system
Correct error	Make a change to correct an error, having localised the cause and knowing how to fix it
Compare example	Compare implementation to an external resource, for example, an online wiring schematic, or program code.
Copy example	Copy or reproduce an external resource, e.g., an online wiring schematic or program code.
Alternative physical input	Use a different input source when interacting with the prototype, e.g., fanning the sensor or blowing on it, to cool it down
Wiggle/push connection	Physically push or wiggle a connection, to make sure it is seated properly or to see the effect upon runtime behaviour/output.
System feedback	Read or use the system feedback printed to the IDE's error panel, for example, compile errors or board communication problems.
Relocate	Move a component or program statement(s) to a different location.
System verify	Compile, to test for correctness
Undo	Reverse a change; usually when a fix attempt is unsuccessful
Reverse orientation	Turn a hardware component around (180 degrees) in the breadboard.
Change power	Change the power source / supply
Control execution speed	Programmatically adjust the speed at which the program is executed (typically by using/modifying the <code>delay()</code> function)
Restart	Restart or reopen something in the hope that this action fixes the problem
Swap for same	Replace a hardware component with one of the same type and specification, for example, to check for a faulty component or test a particular sub-circuit
Cross-check	Visually check that things match where they are supposed to, for example, pin number used in the circuit and program, or baud rates in the program and IDE
Measure	Measure some aspect of the circuit with a specific tool, for example, a digital multimeter
Press ahead, regardless	Stop troubleshooting and return to building, even though there is some suspicion—or even certainty—that there is still a problem

4.2.7 Bugs

To determine whether participants' behaviour was effective, I analysed the coded dataset in conjunction with the bugs and fixes coding from the previous analysis (Chapter 3), looking at whether they successfully resolved all their circuit bugs and how many bugs (of all types, not just circuit bugs) they fixed and introduced when troubleshooting circuit bug-related problems.

4.3 Results

Table 11 shows a summary of the troubleshooting analysed. Of the fourteen participants who troubleshooted circuit bugs, only two participants succeeded in completing the task, successfully resolving all their circuit (and program) bugs, while a further two participants who failed the task, *did* manage to resolve all their *circuit* bugs but not all *program* bugs. The remaining ten participants in this analysis failed the task primarily due to their inability to resolve circuit bugs that they had introduced, either prior to the troubleshooting in this analysis, or during it. The table also details the number of runs, the number of episodes and the number of bugs (all or circuit bugs) introduced or fixed during the troubleshooting of circuit bugs, and the amount of time spent on this troubleshooting. As the event type coding was used primarily to facilitate subsequent segmentation (episodes) and coding (tactics and activity types), rather than as a measure of performance, I do not report the number or total of event type code instances.

I will now present my findings in two parts. First, I analyse *how* participants troubleshooted their circuit bugs, examining where their activity was focused and the tactics which they employed (RQ1). Then, I identify which tactics were *effective* for troubleshooting, looking at their outcomes and comparing participants who successfully resolved their circuit bugs with those who did not (RQ2).

Table 11. Summary of participants' troubleshooting of circuit bugs and the outcomes thereof

Ptc	Task success	All circuit bugs resolved?	No. of runs	Time spent	No. of episodes	Bugs added (all)	Bugs fixed (all)	Circuit bugs added	Circuit bugs fixed
P01	N	N	2	00:20:56	34	10	1	1	1
P02	N	N	2	00:29:15	86	28	18	15	11
P04	N	N	2	00:14:53	27	2	2	2	2
P08	N	N	2	00:06:38	21	4	1	0	0
P10	N	N	2	00:07:26	22	7	5	5	5
P11	N	N	1	00:04:04	3	1	3	1	3
P12	N	N	1	00:06:37	8	1	0	1	0
P13	N	Y	1	00:02:36	12	4	4	4	4
P14	N	N	2	00:25:18	60	12	6	1	1
P15	N	N	2	00:05:58	29	3	3	3	3
P16	N	Y	2	00:01:50	9	1	3	1	3
P17	Y	Y	1	00:15:16	47	14	16	8	10
P18	Y	Y	1	00:02:44	11	10	13	6	9
P20	N	N	5	00:20:32	70	23	15	11	11
			26	02:44:03	439	120	90	59	63

4.3.1 How do end-user developers troubleshoot circuit bugs? (RQ1)

4.3.1.1 Activity Types

I coded each troubleshooting episode with one or more of the three *activity types*: Diagnose, Fix, and Evaluate Fix—which I considered to be the three primary goals of troubleshooting activity within the iterative troubleshooting cycle. Figure 30 shows the number of episodes of each activity type coded for each participant.

Across all participants, 251 of the 439 episodes (57%) were spent trying to *Diagnose* problems, while *Fix* was coded in 130 episodes and *Evaluate Fix* occurred 133 times (Table 12). Some episodes contained more than one activity type (see Figure 31), for example, 70 episodes were coded with both *Evaluate Fix* and *Diagnose* (28% of all 251 *Diagnose* episodes, and 53% of all

133 *Evaluate Fix* episodes). In these episodes, participants were often unsure as to whether a fix had been successful, and diagnosis was usually embedded in their evaluation of this.

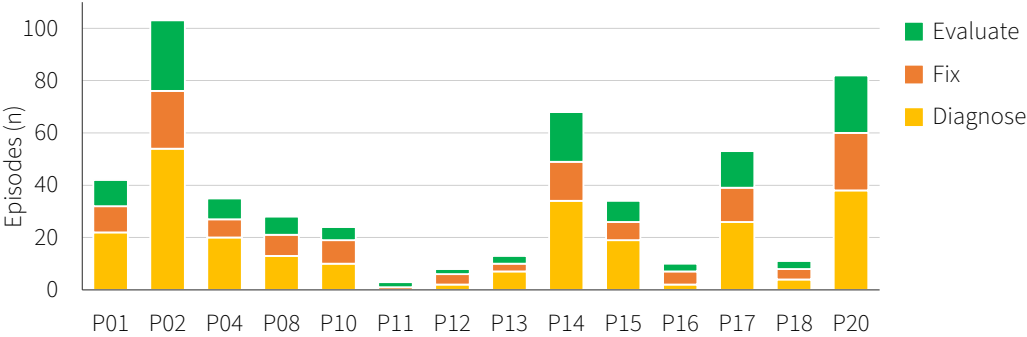


Figure 30. Number of Diagnose, Fix and Evaluate Fix episodes, per participant

Recall that the *Diagnose* activity type code was applied to episodes where a participant knew or suspected that they had a problem and showed evidence of trying to diagnose it—for example, to confirm a problem, to explore or understand a problem, or to identify the cause of a problem. Over half of participants’ troubleshooting episodes involved these types of diagnosis activities (251/439) and they spent almost twice as many episodes trying to identify and localise their problems as fixing them (251 *Diagnose* episodes; 130 *Fix* episodes).

Table 12. Activity Type episode counts, and the percentage of all (439) episodes these represent

Diagnose	Fix	Evaluate Fix
251	130	133
57.2%	29.6%	30.3%

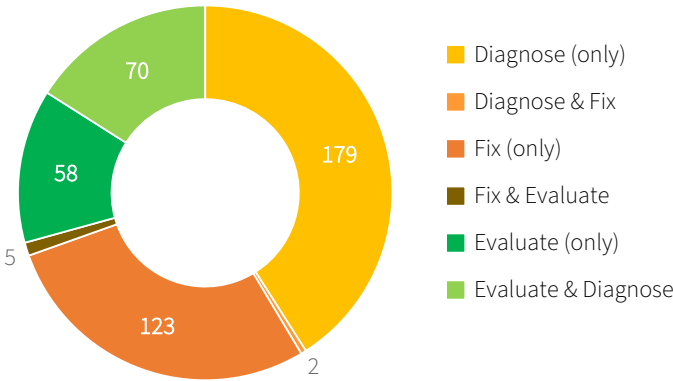


Figure 31. Activity type episode counts, also indicating episodes where more than one Activity Type was coded, most notably where participants were evaluating and diagnosing (70 episodes), i.e., they were unsure if a fix had been successful

Next, I analysed the transitions between these activity types, that is, a move from one type of activity to another, or a particular sequence of moves. An idealized troubleshooting process, according to the general model presented in (Katz and Anderson 1987), is a sequence of transitions from *Diagnose* (locate error) to *Fix* (repair error) to *Evaluate Fix* (test system), with an assumption that a transition from *Diagnose* to *Fix* happens once the person troubleshooting has a clear idea what the problem is, i.e. they have located the cause of it. This idealized sequence occurred 93 times across all participants. There were frequent transitions between *Diagnose* and *Fix* (99) and between *Fix* and *Evaluate Fix* (107), as might be expected from the idealized troubleshooting model.

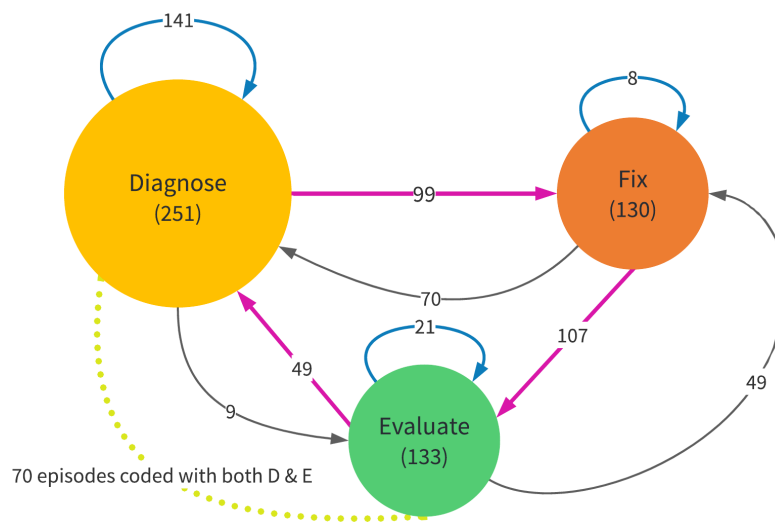


Figure 32. Transitions between troubleshooting activity types

Participants iterated a great deal *within* activity types, particularly within *Diagnose* (141), compared to within *Evaluate Fix* (21) and *Fix* (8). These iterations can indicate that a particular tactic has not achieved the intended goal, and that it either has to be repeated or a different tactic employed. For example, in a particular series of *Diagnose* episodes, P20 performs the following sequence of actions:

1. Visually inspects the circuit, checking the wiring
2. Switches focus to an online example image of sensor wiring, comparing it to their circuit.
3. Reopens the Serial Monitor and interacts with the sensor, observes that the readings are too high and that transitions do not appear to be correct
4. Visually cross-checks the baud specified in the program and the baud specified in the Serial Monitor window, to determine whether they are the same.

5. Turns off Auto-scroll in the Serial Monitor and interacts with the sensor while watching the readings, noting that they do not change as expected.
6. Visually cross-checks the analog pin number referenced in the program with that used on the Arduino board, making sure they are the same.
7. Visually inspect the sensor wiring again and notices that the Ground wire is seated in the wrong breadboard hole.

I would not suggest that the above sequence of episodes is an example of poor troubleshooting. It does, however, serve to illustrate how the number of iterations within Diagnose (141) demonstrate that diagnosis tactics were often not successful in localising a circuit bug and did not lead directly to a fix attempt. Furthermore, as I describe in the next section, some participants transitioned from Diagnose to Fix and made changes to the circuit even though they had not been able to successfully diagnose the problem, i.e., locate the error that was causing it.

4.3.1.2 Tactics

I now turn to how participants employed different troubleshooting *tactics* in each of the activity types. I focus on the most frequently adopted tactics and some less frequently observed which I perceived to have impact (positive or negative) on participants' troubleshooting, or are specific or unique to physical computing development or circuit troubleshooting. I provide examples of participant behaviours that were coded by these tactics and in how many episodes each tactic was observed across all participants.

Tactics observed in Diagnose episodes

Tactics coded within this activity type were characterized by the gathering of information, the processing of that information in order to reach some conclusion about a problem, or some understanding of the conditions under which a problem did or did not manifest. Episodes were coded with more than one tactic, if appropriate.

The most frequently observed tactic was *Run & Analyse* (116 Diagnose episodes; 46%), where participants attempted to use the runtime behaviour of their prototype (for example, the LEDs turning on/off) or output from it (for example, readings from the temperature sensor printed to the Serial Monitor) to diagnose their problem. Participants were observed trying different

approaches to affect the runtime behaviour of their prototypes, for example, changing the temperature sensor's readings by blowing on it, fanning it with a piece of paper, or swapping the hand with which they held it—*Alternative Physical Input* was coded for these 17 episodes (7% of all Diagnose episodes). The *Causal Reasoning from Output* tactic (56 Diagnose episodes; 22%) was coded where participants analysis of runtime output or behaviour resulted in verbal hypotheses—whether correct or not—about the cause of their problems.

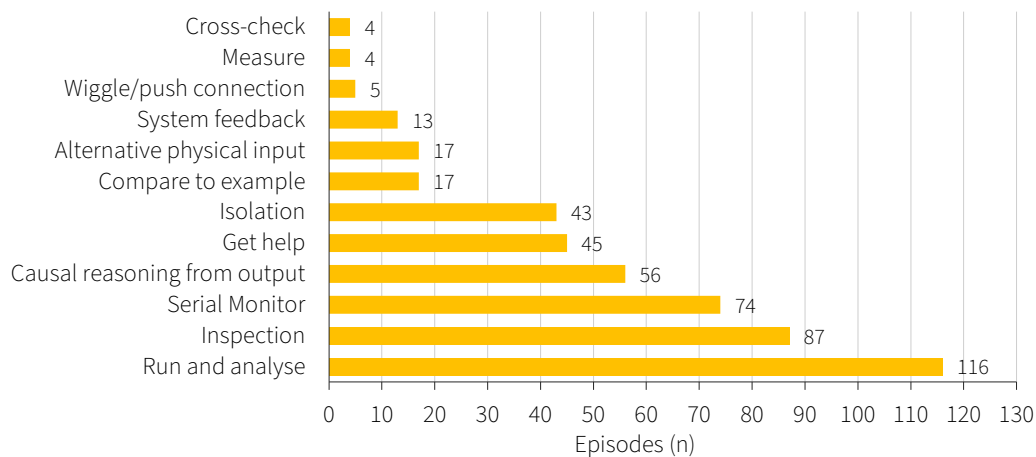


Figure 33. Tactics observed in Diagnose episodes

Participants sometimes used tools to diagnose the runtime output or state of their prototype. In 74 Diagnose episodes (29%), participants analysed output printed to the IDE's built-in *Serial Monitor*, for example, the temperature readings from the temperature sensor. In 13 episodes (5%), they tried to use system feedback in the IDE's error panel to diagnose their problems—while feedback was often related to program errors, some circuit errors, for example, those relating to the connection between the Arduino board and the computer, resulted in error messages. Another tool-related tactic, less frequently observed but particularly relevant to circuit problems, was *Measure* (4 Diagnose episodes, 2%), where participants used a digital multimeter to measure some aspect of the circuit at runtime, for example, voltage.

Sometimes, just having a good look at something can help to identify an error, or rule one out, thereby narrowing down the list of potential causes. The second most frequently observed Diagnose tactic was *Inspection* (87 Diagnose episodes; 35%), where participants visually inspected some aspect of their program, circuit, or IDE configuration, for example, checking that they had connected an LED to the correct Arduino pins, or that it was seated the right way around in the breadboard. *Cross-check*, a specific type of inspection tactic, was observed in 4 Diagnose episodes (2%)—this involves checking that references correspond, for example, that

the digital pin numbers referenced in the program (software) match the numbers (labels) of the digital pins used on the Arduino (hardware).

Participants did not always rely on their own knowledge to diagnose their problems. *Get Help* was observed in 45 Diagnose episodes (18%) where participants looked for and/or used external help to diagnose their problems, for example, they attempted to find information to explain the symptoms they were seeing. Similarly, in 17 Diagnose episodes (7%), participants employed *Compare to Example* as a tactic, where they compared what they had implemented to an external resource, for example, a circuit diagram online, in the hope of identifying any mistake they might have made which could be causing their current problem.

Another tactic observed was to reduce the complexity of the system. I coded *Isolation* against the 43 Diagnose episodes (17%) in which participants were observed attempting to reduce system size or complexity as a tactic for homing in on the cause of a problem, or performing some sort of test upon a reduced system, for example, using direct connections from power and Ground to test whether LEDs were working.

Finally, *Wiggle/Push Connection* (5 Diagnose episodes; 2%), was coded when participants speculatively prodded wires or components at runtime to see whether that changed the behaviour of the circuit, for example, if an LED lit up, or the values printed to the Serial Monitor changed.

Tactics observed in Fix episodes

Two very distinct tactics accounted for 126 of the 130 Fix episodes. The tactic used most frequently when participants attempted to fix their problems was *Speculative Change* (89 Fix episodes; 68%), where participants made it clear that they had not located the error or where there was evidence of speculation or a lack of confidence either in what the error was or that the change they were making would fix the problem. In contrast, 37 Fix episodes (28%) were coded as *Correct Error*, which involved participants carrying out a fix with a clear indication that they *did* have a good idea of what was causing their problem and/or had confidence that what they were attempting would help to rectify it. Some Fix episodes were also coded with a second tactics code, and I now describe these.

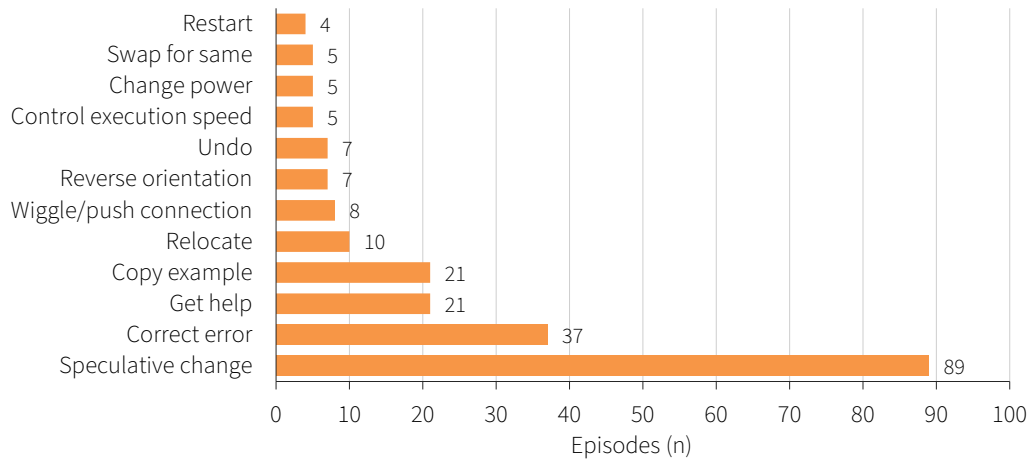


Figure 34. Tactics observed in Fix episodes

In 16% of Fix episodes, participants used external resources in fixing (or attempting to fix) their prototype—the codes *Get Help* and *Copy Example* were applied to these episodes, which usually involved a participant reproducing a circuit wiring image/diagram, or copying code found online. However, almost half of these episodes (11/21) were also coded with *Speculative Change*—even when copying examples participants were not always sure that they were doing the right thing.

I observed a number of other tactics that participants employed in the hope of fixing their circuit bugs. *Relocate*—for example, unnecessarily moving a component to a new position on the breadboard, or the end of a wire to a different position within the same breadboard row—was used in 10 (8%) of the Fix episodes. When using the *Wiggle/Push Connection* tactic in Fix episodes (8 episodes, 6%) participants pushed components or wires deeper into the breadboard or an Arduino pin. *Reverse Orientation* (7 Fix episodes, 5%) was sometimes used to determine whether an LED had been placed the wrong way round in a breadboard—a common mistake when constructing electronic circuits. *Undo*—reversing a fix attempt, for example, if it was unsuccessful (or felt to be unsuccessful) in resolving the problem—was coded against 7 Fix episodes (5%). *Control Execution Speed*—changing `Delay()` values in the Arduino code, primarily to change the rate at which the sensor was read—was seen in 5 Fix episodes (4%); *Change Power* (5 Fix episodes, 4%) usually involved changing the sensor’s power supply from one voltage to another. *Swap for Same* (5 Fix episodes, 4%)—involved swapping out a component for an identical one; and, finally, *Restart*—pressing the restart button on the Arduino board, or reuploading the program without having made any changes—was seen in 4 Fix episodes (3%).

Tactics observed in Evaluate Fix episodes

In most of the 133 Evaluate Fix episodes, the tactics used involved looking at runtime behaviour to see if it met the specification in the brief. Consequently, as with the Diagnose activity type, *Run and Analyse* (108 Evaluate Fix episodes; 81%) was the most frequently used tactic, and the *Serial Monitor* (60 Evaluate Fix episodes; 45%) and *Alternative Physical Input* (14 Evaluate Fix episodes; 11%) tactics were also frequently employed. Many tactics for diagnosing problems are also applicable for evaluating fixes, demonstrated by participants' use of *Inspection* (11 Evaluate Fix episodes; 8%), and *Get Help* (5 Evaluate Fix episodes; 4%) combined with *Compare to Example* (5 Evaluate Fix episodes, 4%), where participants used an external resource to determine whether a fix had been correctly implemented. Participants also used direct connections from 5V power to test whether LEDs were correctly wired within the circuit, i.e., a form of *Isolation* (9 Evaluate Fix episodes, 7%), temporarily removing other dependencies, for example, program logic.

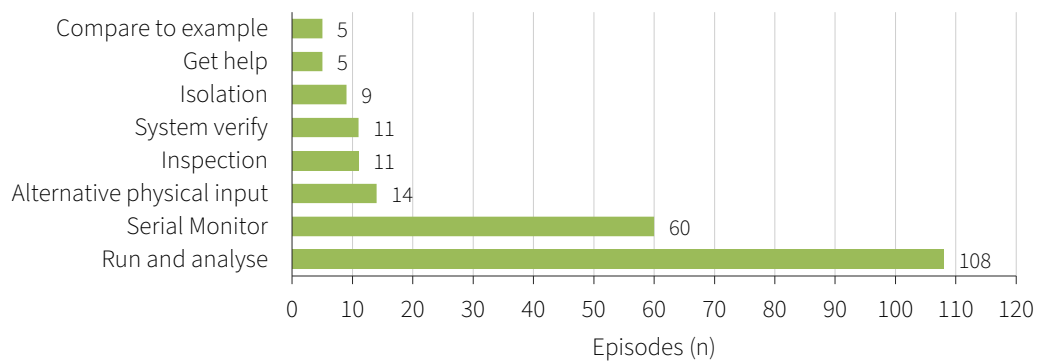


Figure 35. Tactics observed in Evaluate Fix episodes

4.3.2 Are end-user developers' troubleshooting behaviours effective? (RQ2)

In the previous section I discussed tactics used by participants when troubleshooting their bugs. I now turn my attention to determining whether these tactics were effective

To gain insight into why the Circuit Success group were successful and the Circuit Failure group were not, I compared the participants who successfully resolved their circuit bugs (Circuit Success group, n=4) with those who were unsuccessful (Circuit Failure group, n=10).

I first analysed the number of episodes spent by the two groups in the three different activity types and there were stark differences: the Circuit Failure group, on average, had more than twice the number of Diagnose episodes and close to twice as many Fix and Evaluate Fix episodes (Figure 36).

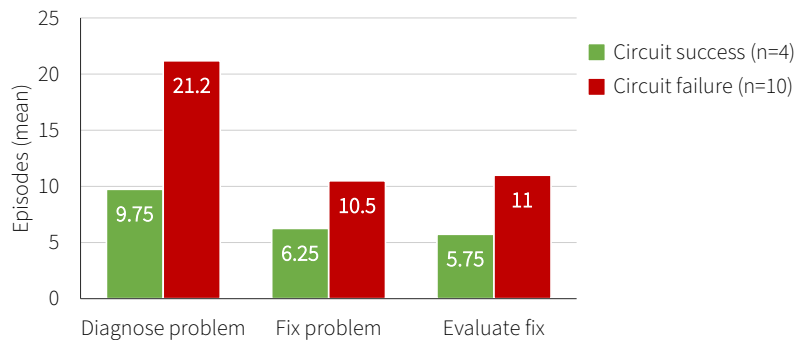


Figure 36. Mean number of activity type episodes per circuit outcome group

I then analysed whether there was a difference between the Circuit Success and Circuit Failure groups in how many bugs they fixed and added when they were troubleshooting (Figure 37). The Circuit Success group fixed, on average, more bugs than they introduced but the Circuit Failure group introduced almost twice as many bugs as they fixed. When I analysed what percentage of episodes had bugs added or fixed, I found that the Circuit Success group fixed bugs in a larger proportion of episodes than they introduced bugs; in contrast, the Circuit Failure group introduced bugs far more often than they fixed them.

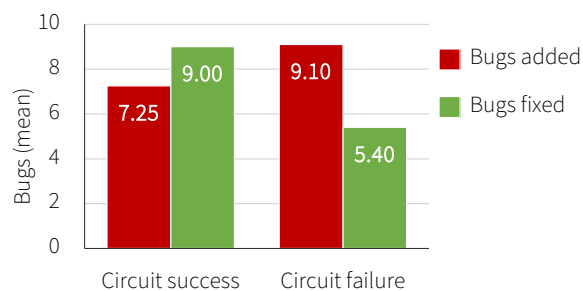


Figure 37. Mean number of bugs added/fixed per circuit outcome group

4.3.2.1 Activity types and tactics

I then analysed the troubleshooting tactics that the Circuit Success and Circuit Failure groups carried out in episodes coded with each of the three activity types, to see if I could identify what made the Circuit Success group more successful troubleshooters.

Diagnose episode tactics

Figure 38 compares the average number of episodes that the Circuit Success and Circuit Failure groups spent carrying out different diagnosis tactics.

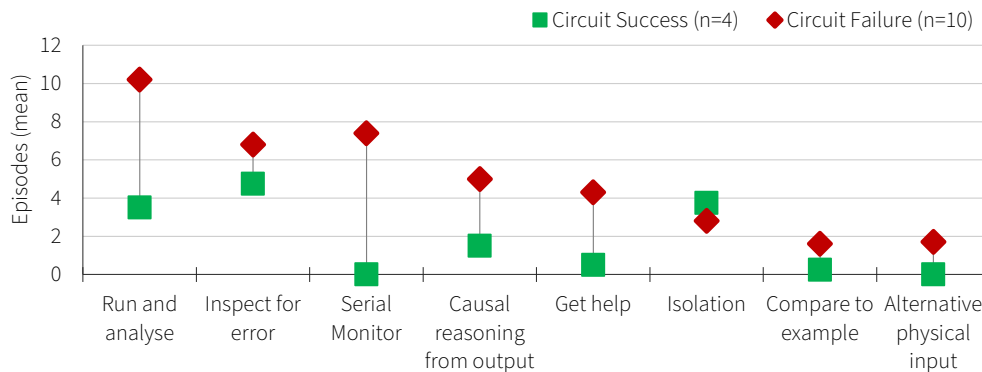


Figure 38. Mean number of episodes coded with each Diagnose tactic used, by circuit outcome group

On average, the Circuit Failure group spent almost three times more *Diagnose* episodes using the *Run and Analyse* tactic, that is, diagnosing their problems through the runtime output or behaviour of the prototype, and on average, made use of the *Serial Monitor* in eight *Diagnose* episodes, in comparison to the Circuit Success group which never used it. They averaged three times more *Causal Reasoning from Output*, episodes, trying to reason backwards from the system behaviour they observed to its possible causes, either watching the behaviour of the circuit, that is, the LEDs, or how temperature readings changed in the Serial Monitor, for example, when they varied the physical input to the sensor—the Circuit Failure group averaged two episodes of providing *Alternate Physical Input* to the temperature sensor, for example, by blowing on it. Analysing behaviour and other output—a ‘backwards reasoning’ strategy (Vessey 1985) can be very useful in diagnosis, however I frequently saw participants in the Circuit Failure group having difficulty understanding the output they were seeing, or generating incorrect hypotheses as to what it might mean. For example, P20, struggling to work out whether the readings they were seeing were normal (they were not) surmised—incorrectly—that the precision of the sensor was the issue, and that this could be addressed by modifying the `Delay()` statement in the program, to control the frequency with which the sensor was read:

"Oh, now it's going down. It takes a while, but, I mean, it works. Perhaps I should try to minimise this effect of the delay we have with the sensor, it's not very precise." (P20)

Similarly, P14 decided to change the temperature thresholds in their program to address the problematic sensor readings they were seeing in the Serial Monitor:

"Hmmm, they're staying at 30. They haven't gone down as much. (Blows gently on the sensor and watches the readings. They don't drop) It's not doing anything. Ah no, now it's going down... no it's staying. That's odd. [...] Yes, it's gone down to 25 again. I'm not entirely sure... what the reading is... 26. [...] Yeah, so those two are too close together, I think. [...] So I might try changing the gap between each temperature" (P14, Circuit Failure group)

In addition, I saw some unsuccessful participants repeatedly running their prototypes without having taken any action that would help them to narrow in on the area of fault, again demonstrating that participants in this group had difficulty using this tactic for diagnosis. Both groups did make some use of the *Isolation* tactic, however, looking at the underlying data, I saw that one participant was responsible for 68% of the Circuit Failure group's *Isolation* diagnosis episodes.

The Circuit Failure group used *Inspection* more frequently than the Circuit Success group, that is, examining their circuits looking for potential bugs, such as miswiring or wrongly seated components, however, it was clear that they did not always know what they were looking for. In some cases, they looked in the wrong location, for example, the program, rather than the circuit, and while some did inspect the circuit, they did not always look at the area of it which contained the bug—again, their hypotheses were poor. For example, P10, had miswired the LEDs, connecting their cathodes to digital pins and their anodes to a power rail. When the LEDs did not light up, they inspected their circuit and incorrectly concluded that the problem lay in the location of the sensor on the breadboard:

"The LEDs aren't lighting up, so that means that I've messed it up. [...] (Inspects the circuit) Oh! But I know why! I need to move this on the other side (pointing at the sensor wiring). I need to move the sensor on this part" (the other side of the central channel of the breadboard)" (P10, Circuit Failure group)

As a result of their inability to localise bugs by analysing output or through inspection, participants in the Circuit Failure group were far more likely to try to *Get Help*, for example, by searching for information that would help them to interpret output, investigate their ideas, or determine whether parts of their prototype were correctly constructed (*Compare to Example*). However, these attempts to find help were not always effective—poor or incorrect hypotheses led to poor or incorrect searches, and in several cases, led participants to make incorrect decisions regarding what might help to resolve their bugs. For example, P14, struggling to resolve their problem, made unnecessary changes to both program and circuit as a result of earlier seeing information online about changing voltage from 5V to 3.3V. Running out of ideas,

they decided to try this, without really understanding it, leading to new circuit and program bugs, further increasing their confusion:

"According to the instructions, it should just be a matter of changing kind of this one value, to tell it it's using a 3.3 volt signal. Um, but I've changed it and now I'm getting negative... (frowns slightly, looking at the program) negative temp... well it's not reading the right temperature, it seems. But I'm not entirely sure where I need to... if there's any other changes I need to make" (P14, Circuit Failure group)

In contrast, the Circuit Success group as a whole only used *Get Help* twice and the *Compare to Example* tactic once. On average, the tactics used most frequently for diagnosis by the Circuit Success group were *Run and Analyse*, *Inspection*, and *Isolation*. The effectiveness of the different diagnosis tactics employed by the Circuit Success and Circuit Failure group become apparent when we compare how they fixed their problems.

Fix episode tactics

I first turn my attention to the outcomes, overall, of the tactics used in *Fix* episodes, as an indicator of whether participants tactics when attempting to resolve their problems were actually successful. Figure 39 shows the outcome of tactics used in *Fix* episodes, in terms of whether they resulted in bug fixes, new bugs, *both* bug fixes *and* new bugs, or neither. We see that some of the tactics most frequently used by end-user developers in this study, not only failed to reliably fix bugs, they actually introduced new ones.

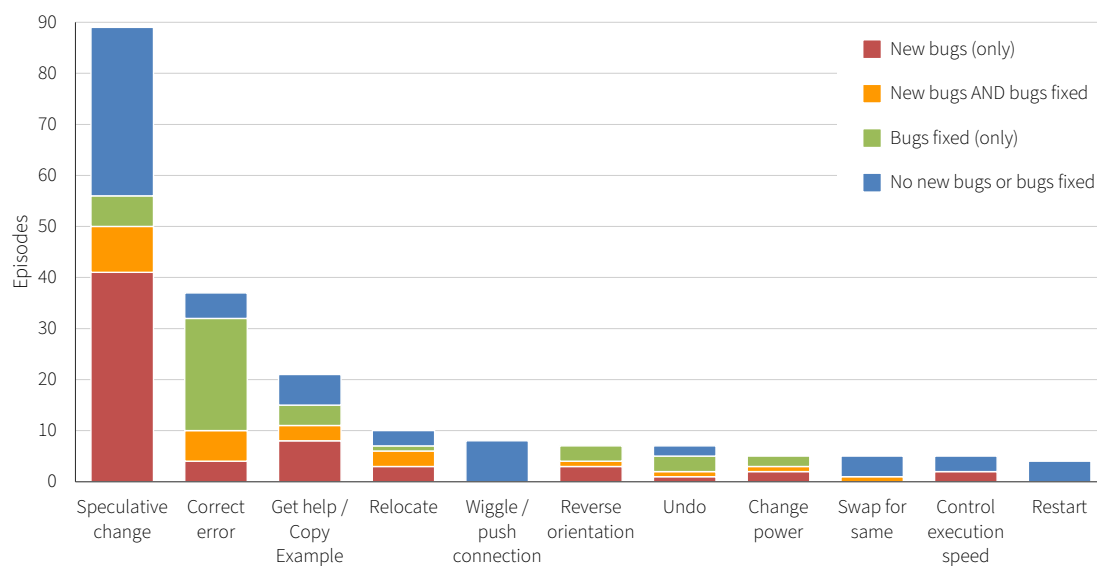


Figure 39. Outcome of tactics employed in Fix episodes, across the whole sample

For example, several end-user developers in this analysis used external resources (*Get Help / Copy Example*—in this activity, both code the same episodes) when attempting to fix their problems, however this was clearly not always effective—participants sometimes made mistakes when copying examples, due to knowledge gaps and poor understanding, or cognitive slips, resulting in new bugs or unnecessary, additional complexity. Overall, eleven of the *Get Help Fix* episodes resulted in new bugs (eight coded with *Speculative Change*), while only seven resulted in bug fixes (two coded with *Speculative Change*) and six of these episodes neither fixed bugs nor introduced new ones, where participants made unnecessary but otherwise harmless changes, for example, minor modifications to the time specified in the `Delay()` statement, to reflect what they had seen in example code online.

Looking specifically at the two main tactics used in *Fix* episodes (Figure 40) we see the *Speculative Change* tactic led to *very few* bug fixes (17% of episodes, 20 bugs fixed) and actually introduced *lots* of new bugs (56% of episodes, 70 bugs added). In contrast, the *Correct Error* tactic did result in a lot of bug fixes (76% of *Fix* episodes, 63 bugs fixed) and far fewer new bugs (27% of *Fix* episodes, 21 bugs added). This reveals an important finding, namely, that the tactic that end-user developers used *most often* to fix their circuit bugs, not only failed to achieve the desired result, but actually introduced more problems.

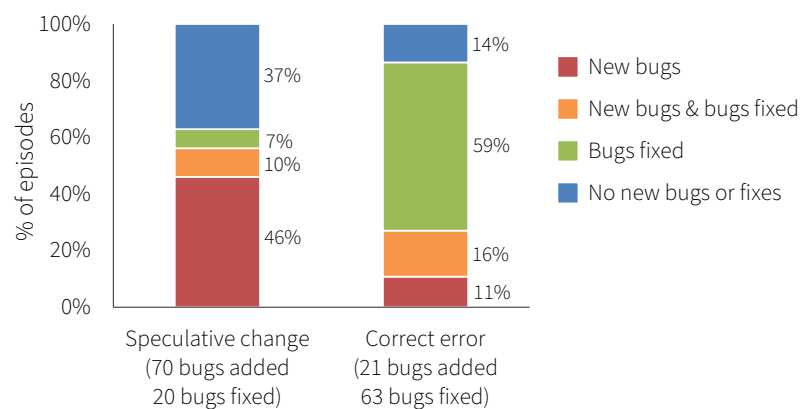


Figure 40. Episode outcomes of the two main *Fix* tactics used by participants

Comparing the two groups provides some insight into why the Circuit Failure group struggled to resolve their problems. The biggest difference between the tactics used by the two groups to fix their circuit bugs is that the Circuit Success group had, on average, almost twice as many *Correct Error Fix* episodes as the Circuit Failure group, who had, on average, over three times as many *Speculative Change Fix* episodes (Figure 41). The Circuit Success group on average employed the *Correct Error* tactic in twice as many *Fix* episodes as *Speculative Change*; in contrast, the Circuit

Failure group on average employed the *Speculative Change* tactic in over three times as many *Fix* episodes as *Correct Error*.

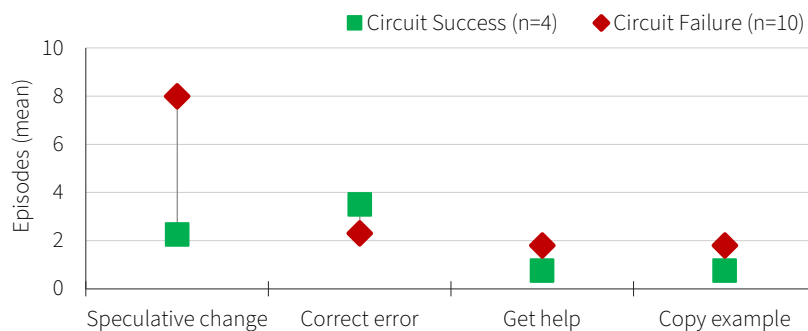


Figure 41. Mean number of episodes coded with each *Fix* tactic used, by circuit outcome group

It appears, from these results, substantiated by the video data, that when the Circuit Success group performed *Fix* actions, they more often had (or appeared to have) confidence that they knew where the error lay and/or how to rectify it, as a result of having effectively diagnosed their circuit problem. In contrast, those in the Circuit Failure group made far more changes without such confidence. The fact that the *Copy Example* fix tactic was, on average, more frequently used by the Circuit Failure group than the Circuit Success group (Figure 41) is also evidence that the Circuit Failure group was less certain about the nature of the circuit bugs they were trying to resolve—they were more likely to *Get Help*. In some cases, participants also did not understand the example they were copying, or struggled to interpret it. For example, P12, when trying to correct their sensor wiring, had difficulty trying to work out how the sensor was oriented in the diagram they found online, with the result that they actually recreated the bug that they were attempting to fix:

"Either it's going to be... (Looks between the board and the image). How can I tell which way round it's going to be, just by looking at it? [...] I may put it in the wrong way again" (P12, Circuit Failure group)

The lack of confidence or certainty that the Circuit Failure group had about the cause of their circuit bugs and how to resolve them was evident in the type of language they used, for example, *"Let's stick that in there and see what happens"* (P01), *"I'll give it a try"* (P10). In fact, the Circuit Failure group often made changes to their program or circuit in the hope that doing so would help them to locate the source of the problem—a trial-and-error or *tinkering* approach to troubleshooting that in many cases led to additional problems.

Evaluate Fix episode tactics

There are also differences in the *Evaluate Fix* tactics adopted by the Circuit Success and Circuit Failure groups (Figure 42). Although both groups used *Run and Analyse* in approximately the same proportion of their *Evaluate Fix* episodes (Circuit Success 83% of 23 episodes, Circuit Failure 81% of 110 episodes), the Circuit Failure group used it, on average, in almost twice as many episodes and made far more use of the *Serial Monitor*—53% of their *Evaluate Fix* episodes, in contrast to the Circuit Success group who only used this tactic/tool twice.

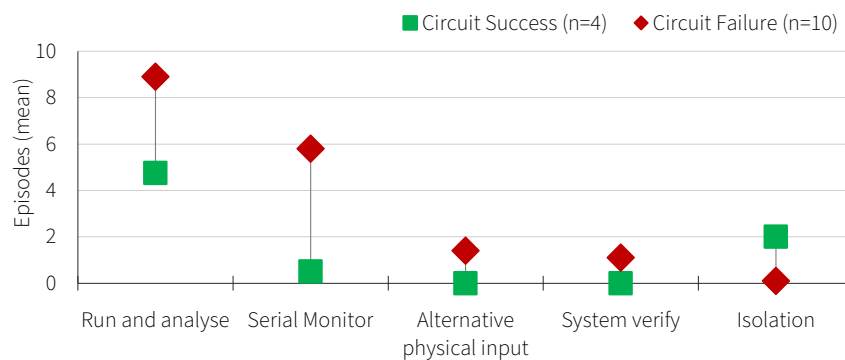


Figure 42. Mean number of episodes coded with each *Evaluate Fix* tactic used, by circuit outcome group

The whole Circuit Success group used *Isolation* to evaluate their fixes in eight episodes (35% of their *Evaluate Fix* episodes), in comparison to the whole Circuit Failure group who only used it once (1%). Both of these tactical differences reflect the fact that the Circuit Success group were more likely—and presumably more able—to evaluate the success of their fix attempts by focusing on a smaller part of the system, compared to the Circuit Failure group, who had to rely more often on program output to the Serial Monitor window.

I will now discuss these results in the context of my research questions and their implications for how to support end-user developers in troubleshooting their bugs.

4.4 Discussion

This study aimed to answer two research questions:

RQ1: How do end-user developers troubleshoot circuit bugs? What troubleshooting tactics do they use?

RQ.2: Are end-user developers' troubleshooting behaviours effective in helping to resolve their circuit bugs?

This analysis demonstrated that the natural troubleshooting behaviours of end-user developers can be very different from the idealised model (Katz and Anderson 1987). While a small number of participants were able to successfully troubleshoot all of their circuit bugs, the majority were not, on average spending roughly double the number of episodes on diagnosing their problems, attempting to fix them, and evaluating the success of their fix attempts.

4.4.1 Troubleshooting tactics

Participants often relied on *analysing runtime behaviour* to diagnose their problems, with limited success when they had difficulty understanding the output they were seeing, or judging its correctness. Successful participants made more use of tactics such as *isolation*—testing or evaluating a smaller functional part of the circuit. Although both successful and unsuccessful participants made use of *inspection* in attempting to diagnose their problems, in many cases, unsuccessful participants applied these visual checks to the wrong location of their prototype. Unsuccessful participants also made use of *external help*, but in several cases, this proved ineffective, particularly when participants' hypotheses were poor or incorrect, affecting their searches and, in turn, the changes they chose to make in the hope of resolving their problems. Additionally, unsuccessful participants also made mistakes when using information found online, for example, when copying example circuit diagrams, or comparing them to what they had implemented.

As we have seen, when participants failed to localise their circuit bugs, they often resorted to making *speculative changes* to their prototype, in the hope that this would fix and thereby reveal the error, however, this approach resulted in over than three times more bugs than it fixed—the majority of this type of fix attempt were made by the Circuit Failure group (80 out of 89 episodes). In contrast, the Circuit Success group were far more successful at localising and resolving their bugs—their fix attempts resolved three times as many bugs as they introduced.

Although the analysis in the previous chapter found no significant correlation between participants' self-rated expertise and either their task success or the problems they experienced (section 3.3.4), there is evidence in the tasks transcripts and videos that participants' lack of success in diagnosing and resolving their circuit bugs, and the further problems they

encountered when doing so, were sometimes related to a lack of domain knowledge, which led to poor or incorrect hypotheses, wrong actions and mistakes. However, it would be wrong to conclude that the participants in the Circuit Success group were all just more knowledgeable and better at troubleshooting than those in the Circuit Failure group. Participants in the Circuit Success group sometimes made speculative changes to their circuits and introduced new bugs; participants in the Circuit Failure group were able to successfully diagnose and fix some, if not all, of their bugs. For example, P02, who was in the Circuit Failure group, fixed more bugs than any other participant (18) but also introduced the most bugs (28). On some occasions P02 demonstrated an exemplary approach to troubleshooting that followed the idealised model (diagnosing effectively, fixing a localised bug and then evaluating that the fix had been successful). However, they were also completely unable to diagnose one particular circuit bug and eventually resorted to making speculative changes which eventually proved fatal. It is informative to compare a bug that P02 solved with the one that led to task failure.

P02 ‘LED orientation’ bug (Resolved): When P02, prior to writing any code, noticed an LED light up unexpectedly, they decided to conduct a test to see if the LEDs were correctly oriented:

"I'm just going to test that the LEDs light up when I connect power to them, so I know they're the right way round. [...] I like to make sure I'm doing the right thing before I get confused and start trying to code around a problem that was my fault in the first place" (P02)

They isolated and tested LED subcircuits by temporarily connecting them directly to the Arduino 5V pin—a direct power source. After conducting the test, P02 hypothesised (correctly) that the LEDs that did not light up were wrongly oriented:

"it doesn't light up. So, therefore these are all the other way round." (P02)

P02 turned those LEDs around in the breadboard and touched each anode with a wire connected to the 5V pin. When each LED lit up in turn, P02 concluded—correctly—that all LEDs were now properly oriented and carried on with the task.

P02 ‘Missing LED resistors’ bug (Unresolved): Unfortunately, when wiring up their LEDs, P02 had failed to add resistors to them. Although this did not prevent the LEDs from lighting up, it meant they drew too much current from the Arduino, which in turn, affected the temperature sensor readings—the insidious problem mentioned in the previous chapter. Unlike the LED orientation bug that they successfully diagnosed and resolved, it was not easy to identify what was causing this problem. P02 carried out a series of exploratory tests and changes to the circuit

in the hope that one of these would reveal the problem, for example, removing sensor connections and wiggling wires, watching whether this changed the readings in the Serial Monitor:

"So, I'm wiggling that wire, and I'm looking at the numbers coming back and they're changing... quite a lot." (P02).

However, observing these readings did not help P02 localise the problem:

"Hmmm, some very weird numbers are coming out of this thing. I wonder if I... I haven't hooked it up backwards, have I? I'll have a look at the documentation again." (P02)

Seeking external help led them to speculatively add an unnecessary resistor to the sensor:

"It's probably not right, but let's see. Hopefully it won't explode" (P02).

This not only did not help them resolve or diagnose the problem, it also actually introduced a new bug, further complicating matters—in fact, they subsequently ended up adding new program bugs, as well as yet another resistor, making diagnosis even more difficult. Leaving the unnecessary resistors in place, and failing to make any sense of the output they were seeing, P02 eventually failed the task.

The 'LED orientation' circuit bug that P02 *did* resolve was less complex to diagnose than their 'missing LED resistors' bug, as they were at an earlier stage in developing the prototype and thus there were fewer dependencies in place—similar to the bugs encountered by the Circuit Success group and some of the bugs that the Circuit Failure group *did* manage to resolve. Three of the Circuit Success group participants were troubleshooting LED wiring bugs, and at that point in development had either not added the sensor to the breadboard or, if they had, they had not yet written the program logic to control the LEDs in response to the sensor readings. It was therefore relatively straightforward for them to deduce that the problem they were experiencing lay in their LED wiring. The fourth participant in this group had only wired up the sensor and written the program to read it when they discovered that temperature sensor was very hot to the touch—it had overheated due to being wired up incorrectly. This provided very clear feedback about the location of the circuit bug. In contrast, when troubleshooting the 'missing LED resistors' bug, although the temperature readings in the Serial Monitor indicated to P02 that there was a problem, they provided no information about the cause or location of the problem. P02's focus was on the erratic behaviour of the sensor, but this was caused by an unobvious dependency on a different component—the LEDs, which, lacking resistors, were

drawing more current. Despite their diagnosis efforts, at no point was P02 able to isolate the cause of the problem and they therefore resorted to making speculative changes, resulting in new bugs.

Beyond end-user developers' domain knowledge and skill in troubleshooting, there are therefore two additional factors that appeared to affect how end-user developers in this study troubleshooted their circuit bugs and whether they were successful at resolving them: 1) the stage of development of the prototype—the less complex the prototype, the easier it was to localise bugs; and 2) the directness of feedback about the location of the bug—hot sensors are easier to diagnose than erratic readings from a temperature sensor that are resulting from hidden dependencies on other parts of the circuit. The Circuit Success group were able to solve all their circuit bugs by diagnosing successfully and making informed changes to the circuit. Whereas all participants in the Circuit Failure group failed to resolve at least one bug because they were troubleshooting a prototype with more complex dependencies and no direct feedback about the location of the bug, which meant they relied on observing the runtime behaviour of the prototype to determine whether there was a bug and if so, where it was located. In some cases, added complexity was due to participants deciding to stop troubleshooting, having failed in their initial diagnoses, to continue with building, making subsequent diagnosis even more difficult. As a result of participants' inability to diagnose their problems and localise their circuit bugs, they made speculative changes. Not only did these mostly fail to resolve their problems, they also introduced more bugs, consistent with the literature on the problems of end-user and novice programmers (e.g., Gugerty and Olson 1986). These insights into the additional factors that influenced how end-user developers tried to troubleshoot circuit bugs in this study, along with other findings from the study, suggest ways that we can support end-user developers in troubleshooting problems in physical computing prototypes.

4.4.2 Supporting end-user developers' troubleshooting

I will now summarise some of the key difficulties I observed in participants' troubleshooting, before reflecting on one of the more problematic behaviours observed—speculative changes—and then outlining types of support from which I believe end-user developers might benefit, based on my findings.

In this study, several participants struggled to diagnose their circuit bug related problems, particularly those in which there were complex or hidden dependencies. A number of ineffective

behaviours were observed, including when participants ran out of ideas for things to try. For example, instead of conducting focused tests, attempting to reduce dependencies, or undertaking systematic inspection, some participants relied on repeatedly running (executing) their prototypes in the hope that this would reveal some clue—more often resulting in poor or incorrect hypotheses regarding the runtime behaviour or output they were seeing, especially if they did not understand what failure/success looked like. When participants did inspect their prototypes for error, they sometimes looked in the wrong place, again due to poor or incorrect hypotheses, while more thorough and systematic inspection, for example, not just of the location in which symptoms occurred, may have led them to localise their bugs. Additionally, when participants sought external help, to diagnose or fix their bugs, or evaluate their fixes, this was not always effective, because their searches were poor/incorrect (due to inadequate or incorrect hypotheses), or they lacked the knowledge to understand, judge or apply the resources they had found. In some cases, instead of attempting to analyse or understand the failure they were seeing, participants did little or no diagnosis before attempting fixes.

Many of these ineffective troubleshooting behaviours resulted in participants making speculative changes to the program and/or circuit, few of which actually fixed bugs. While some instances of good/effective speculative changes were observed, most speculative changes introduced far more bugs than they solved. Participants sometimes left these bugs in place, rather than undoing them, which compounded their difficulties. They also added to their problems by choosing to make several changes in one go, rather than incrementally making and immediately testing changes.

4.4.2.1 Speculative changes—*tinkering*

Tinkering, as a pattern of behaviour, has been observed in novice and end-user programmers (e.g., Perkins et al. 1986; Cao et al. 2010) and it may even be that end-user developers in the current domain are particularly prone to it, due to the hands-on nature of circuit construction, and of making in general. Tinkering as a creative approach to making is part of Arduino's philosophy, where people who are not experts in programming or electronics are encouraged to 'have a go', and to discover and learn from their hands-on experiences, including their mistakes (Banzi 2009). However, this study suggests that a 'try it and see what happens' approach *without* thinking or reflection is, in itself, a poor troubleshooting strategy. There is therefore an interesting tension between the informality of tinkering as a form of creative experimentation,

and the potential for a more structured, systematic approach to troubleshooting that could help end-user developers to locate their bugs and resolve their problems.

It is not only unrealistic to expect end-user developers *not* to make *any* speculative changes when troubleshooting, there can be *value in experimentation*, both in terms of helping to diagnose and resolve problems within a troubleshooting process, and as opportunities for end-user developers to learn through their experiments—other researchers have pointed out both negative *and* positive effects of novices' tinkering when programming (Perkins et al. 1986). I did see instances of what we might call 'good'—or what Perkins and colleagues would term *effective*—tinkering. These instances were often characterized by small, low-cost changes (in terms of time and effort) that were easy to evaluate, easily reversible, and which were unlikely to have adverse effects. When making these changes, participants often also seemed to have a clear idea of what outcome they were looking for—for example, P13 turned an LED around in the breadboard, to see if it was wrongly oriented, and then immediately turned it back when it did not light up. In this form, tinkering can be viewed more as *focused hypothesis testing*. Speculative changes in this study often worked best when fewer dependencies existed, either due to the stage of prototype development or to participants deliberately isolating the part of the system in which to test a potential hypothesis. In a nutshell, it seems that if end-user developers are more thoughtful when making speculative changes, they could avoid some of the additional problems introduced by participants in this study when troubleshooting. Equally, if end-user developers are better at diagnosing their problems through other tactics, this may also result in fewer speculative changes of the kind we saw the Circuit Failure group resort to when they ran out of ideas.

Therefore, a general troubleshooting strategy that would encourage end-user developers to adopt a more considered and less speculative approach to troubleshooting is *more thoughtful/reflective troubleshooting*—thinking before, during and after action.

4.4.2.2 Support for specific aspects of troubleshooting

Chiefly, this study demonstrates that end-user developers would benefit from support for *diagnosing* their problems, including complex circuits that contain multiple dependencies. If end-user developers are better at diagnosis, they are more likely to localise their bugs and therefore make more *informed* changes to their prototypes, rather than haphazard speculative ones. Furthermore, if end-user developers have the ability to diagnose their problems, this will

also help them to evaluate the success of their attempted fixes too, as both activities employ some similar tactics.

The study suggests that specific aspects of diagnosis/evaluation that would benefit from support include:

- ***Planning and hypothesising:*** It would be unrealistic to expect end-user developers, particularly novices, to formulate complex plans upfront before acting, however, considering different hypotheses, and troubleshooting tactics, weighing up options, and making more thoughtful decisions regarding what action to take, should help end-user developers to become better troubleshooters.
- ***Recognising and defining failure:*** Better identification and analysis of the symptoms caused by bugs could help end-user developers with problem diagnosis and fault localisation, but also in evaluating whether their attempted fixes have been successful. End-user developers should be encouraged to look for and define symptoms of failure, in order to generate better hypotheses.
- ***More focused analysis of runtime behaviour/output:*** End-user developers could perform more focused analysis of the runtime behaviour and output they can observe, rather than just repeatedly running (executing) their prototypes and hoping that this will reveal some clue. End-user developers would benefit from guidance in the types of analyses they can perform.
- ***Problem decomposition:*** Breaking a problem down or simplifying it can aid in diagnosis and fault localisation. Reducing dependencies, through tactics such as isolation, can help to establish the boundaries of failure and identify which elements of a circuit or program are contributing—and not contributing—to failure. End-user developers would benefit from suggestions to adopt these approaches.
- ***Focused testing, rather than haphazard speculative changes:*** There is value in experimentation, however, potential risks and dependencies should also be considered. Ideally end-user developers should be more thoughtful when making speculative changes, approaching these as focused tests driven by hypotheses, ideally with a clear idea of what to look for in the results. This would also make it easier for end-user developers to evaluate the results of their experiments and reverse changes if needed.

- **Thorough inspection:** A number of visual checks should be performed before making changes. Suggestions of where to look and what to look for—particularly common errors—could help end-user developers to localise their bugs.

End-user developers can also be supported by providing best practice for making changes to circuits, for example:

- **Incremental, iterative progress:** End-user developers should be encouraged to make one change at a time and evaluate the results before making further changes.
- **Keeping track:** Having a record of what was tried and what the results were could prevent end-user developers from repeating unsuccessful troubleshooting actions.
- **Undo failed fixes:** End-user developers should be encouraged to reverse changes that did not fix their problems, rather than building further upon them.

Finally, end-user developers should be encouraged to **follow an iterative process**—performing thorough diagnosis before a fix attempt, and then immediately evaluating whether the fix was successful.

4.4.2.3 General troubleshooting support principles

There are three over-arching principles to support end-user developers' troubleshooting:

First, as suggested, we should encourage end-user developers to **be more thoughtful/reflective when troubleshooting** and to avoid making speculative changes that typically result in more new bugs than fixes. Dewey (discussed in Miettinen 2000) distinguished this type of trial-and-error behaviour from *reflective problem solving*, which he saw as an iterative cycle of: defining the problem; diagnosing and formulating a working hypothesis; reasoning; and testing the hypothesis through action. Fleck and Fitzpatrick (2010) define reflection as “*serious thought or consideration*” and identify five levels of reflection (R0-R4), where the lower levels are foundation for higher ones. The card-based support tool I describe in the next encourages end-user developers to reflect at the *R2* level—*Dialogic Reflection*—that is: thinking about what they should be doing and why, considering alternatives, questioning their assumptions, generating and prioritising hypotheses to explore, and evaluating their fix attempts and the impact of their actions. In general, thinking through their problems should help end-user developers become better troubleshooters and increase the likelihood of them solving their own problems.

Second, support should facilitate end-user developers *persevering with systematic troubleshooting*. In this study, some participants gave up troubleshooting and continued building when they were not sure that they had solved their problem, or when they ran out of ideas.

Third, end-user developers would benefit from support in *planning and tracking their troubleshooting*. This would help them carry out all necessary steps and also enable them to remember what they have tried and what the results were.

In summary, I suggest that end-user developers can be supported in troubleshooting by providing them with alternative tactics to use, based on the recommendations for specific support I have outlined, and underpinned by some general principles, for example, encouraging end-user developers to be more thoughtful when troubleshooting. The aim is not to try to turn end-user developers into engineers, as a big appeal of physical computing is its creative, tinkering approach to making interactive devices. However, with this support I feel a number of unproductive—and in some cases, destructive—troubleshooting behaviours observed for some of the participants in this study could have been avoided.

It is worth noting that while the above recommendations are made in respect to supporting end-user developers' troubleshooting within a physical computing development context, most of these recommendations are reasonably high-level and would potentially be applicable to other development domains.

Finally, as with the previous chapter, some limitations should be considered in respect to the findings in this study. Threats to validity include, for example, the small sample size, and representativeness of this sample of the wider end-user developer population, but also the potential effects of the study itself, for example time constraints and the pressures felt as a result of being observed and recorded. I discuss these and other limitations in section 7.2.

In the following chapter, I will describe the design and development of a novel support tool which instantiates the suggestions for supporting end-user developers that are summarised above, in a physical card-based form.

Chapter 5

Developing a physical card-based tool to support end-user developers' troubleshooting

5.1 Introduction

The previous two chapters describe my empirical work investigating how end-user developers develop physical computing prototypes and troubleshoot circuit bugs which they introduce. These analyses enabled me to identify ways in which end-user developers could be supported, summarised in section 4.4.2.

Informed by these studies, I now describe the design and development of a novel, card-based tool to support end-user developers' troubleshooting of physical computing problems, particularly circuit bugs, addressing the third main research question of this thesis:

TRQ3: How can we design a deck of physical cards to support end-user developers in troubleshooting physical computing problems, particularly circuit bugs?

The structure of this chapter is as follows: I first establish a context for the development of this new support tool, in comparison to previous work to support end-user developers. I then motivate the use of cards as a medium in which to provide support, and describe the design principles which guided the design and development of the card deck. I describe the development of an initial prototype and how trialling this in a small proof of concept study led to targeting the design of the tool more towards needs of novice end-user developers. Thereafter, I report the findings of two focus groups with novice end-user developers, to help shape the design of the final card set. Finally, I describe the card set that was used in the final study of this thesis, along with some supplementary materials in the toolkit, and detail the production process.

5.2 Designing cards to support troubleshooting

As discussed in my review of the literature (section 2.8.1), there has been much work to find ways to support end-user programmers in debugging their software programs. For example, software tools such as the Idea Garden (Cao et al. 2015; William Jernigan et al. 2017) draw upon theories of learning and curiosity (Carroll and Rosson 1987; Robertson et al. 2004) to provide in-situ support for end-user programmers during their tasks, and help them to become better problem solvers. Tools such as these rely on background analysis of the user’s program, and have been shown to be effective, but only address programming issues. As the empirical work in the previous chapters demonstrates, end-user developers do experience problems with programming their physical computing prototypes, but *circuit*-related problems appear to have the most significant impact upon task success, with some circuit bugs proving particularly difficult to diagnose and resolve. Since that work, research prototypes have been developed by other researchers to help learners debug electronic circuits (section 2.8.2), however, with the exception of Bifröst (McGrath et al. 2017) these deal only with electronic circuits, not programming, and again rely on automated analysis of implementation, often requiring additional hardware and software. They also do not address some of the additional support needs I identified in the previous chapter. I am aware of no other tool aimed at end-user developers that, in addition to supporting the diagnosis of circuit bugs, specifically aims to scaffold the *process* of troubleshooting physical computing problems and help novice end-user developers to become better problem solvers in general. In the face of *fragile* knowledge, such as that demonstrated in the troubleshooting difficulties of participants in the previous chapter, it has also been suggested that simple prompts may perhaps play a better role than sophisticated strategies, in helping novices to think through their problems and build their knowledge effectively (Perkins and Martin 1986)—something worth considering for end-user developers too.

As my empirical work suggests that end-user developers would benefit from applying more *process* and *reflection* to their troubleshooting activities, I looked to other domains for inspiration when considering through which medium to support might be provided. A popular method used to generate ideas, prompt reflection, and provide low-tech support for process in other domains, either in general, or for particular activities, is *physical cards*.

5.2.1 Why cards?

Numerous card-based tools have been developed to support the generation and development of ideas within a creative or design process and/or to provide theoretical, domain-specific knowledge during one. Domain, problem or activity-specific card tools include:

- *Tango Cards*—designing tangible learning games (Deng, Antle, and Neustaedter 2014),
- *Mixed Reality Game Cards*—designing mixed reality games (Wetzel, Rodden, and Benford 2016),
- *Exertion cards*—designing exertion games, (Mueller et al. 2014),
- *DSD Cards*—instantiating knowledge useful when designing technology for children (Bekker and Antle 2011),
- *Tangible Interaction Cards*—designing tangible user interfaces (Hornecker 2010),
- *Information Privacy by Design Cards*—considering data protection issues during design (Luger et al. 2015),
- *PLEX Cards*—designing for playfulness (Lucero and Arrasvuori 2010),
- *Video Card Game*—supporting User-Centred Design discussions (Buur and Soendergaard 2000),
- *The Design with Intent Toolkit*—designing for behaviour change (Lockton et al. 2009)
- *Envisioning Cards*—considering human values during design (Friedman and Hendry 2012)
- *VNA*—game design ideas (Kultima et al. 2008) and
- *Tiles IoT Toolkit*—designing Internet of Things prototypes (Mora, Gianni, and Divitini 2017).

Card tools have been shown to afford a number of benefits. Antle and Wise, using theories of cognition and learning to inform TUI design, argue that “*Using spatial, physical, temporal or relational properties can slow down interaction and trigger reflection*” (Antle and Wise 2013), suggesting that physical cards, which end-user developers can easily and rapidly arrange to explore relations, and configure into meaningful spatial arrangements, are an appropriate medium for encouraging reflective troubleshooting. Bekker and Antle also found that cards were good for facilitating comparisons as they can be rapidly moved next to one another or formed into groups (Bekker and Antle 2011). Furthermore, if there are different categories of cards, they

can be combined in different ways to explore new possibilities. More generally, arranging cards can support the framing and reframing of a problem and lead to the generation of hypotheses—something shown to be problematic for novices (Vessey 1985; Gugerty and Olson 1986). Having duplicate cards can support the generation of alternative hypotheses (Buur and Soendergaard 2000) and the use of cards containing less-specific information can also help generate ideas and hypotheses (Wetzel, Rodden, and Benford 2016). Cards can not only embody or instantiate knowledge that will be useful during a process, they can also act as good memory prompts (Deng, Antle, and Neustaedter 2014), of relevant information, but also of where a user is in the current process, for example, they can be used to help break down a problem into individual steps and prevent important steps being missed out (Mueller et al. 2014).

5.2.2 Design review of existing card sets

To gain insight into designing a card-based tool, I looked to the academic literature—including but not restricted to the design, creativity, HCI and education literature—and also at tools originating outside of academic research, for example commercial card decks. The intention was not only to gain an understanding of the tools available and how the design of these tools supported their stated aims and uses, but also to uncover the different factors important in the process of designing a card-based tool, and delivering information via this medium.

Additionally, I found suggested classifications of ideation card tools, which also informed the design process. For example, Wolfel and Merritt (2013) did a comparative analysis of eighteen card-based tools and described them in terms of five design dimensions: Intended purpose and scope; Duration of use / When in process; Methodology of Use; Customization; and Formal Qualities. More recently, Roy and Warren (2018) list several ways in which card sets can work within a process, for example, to stimulate creative thinking (e.g. Brian Eno’s Oblique Strategies cards ‘The Oblique Strategies’, n.d.), summarise useful information and knowledge (e.g. The Game Design Deck of Lenses, Arcila 2013) or to provide ideas and/or solutions for specific contexts or domains (e.g. the Design with Intent toolkit, Lockton et al. 2009).

The next section presents my findings about the *design* of physical card-based tools, drawn from the academic literature.

5.2.3 Considerations when designing cards

Based on my review of the literature, I identified five key categories of design considerations:

Physical form

The physical format chosen for a set of cards should take into account how they will be used, considering, for example, their handling and positioning. Physical properties, such as the size and thickness of cards, have been shown to matter: participants in a study to evaluate the DSD cards (Bekker and Antle 2011) felt that sturdier materials would have improved the cards' tangible properties; similarly, in a study by Tudor et al., cards' lack of stiffness affected how participants used them. These authors also found that over-sized cards were difficult to manipulate (Tudor et al. 1993). The orientation of card design can affect the readability of content when cards are used by groups (Buur and Soendergaard 2000), but orientation also has implications for handling and positioning by individual users too, as does sidedness—usually only one side of a card can be seen unless the user turns it over. Most cards typically use a rectangular format, but different shapes may afford ways to indicate differences or relationships between cards (Bekker and Antle 2011).

Information content

The information on a card should support its purpose and reinforce desired behaviour. When designing cards to prompt thinking or reflection, questions—particularly open ones—are one mechanism commonly used towards this end (Lockton et al. 2009; Hornecker 2010; Bekker and Antle 2011; Friedman and Hendry 2012; Mueller et al. 2014; Luger et al. 2015); others include providing minimal information (Kultima et al. 2008; Lucero and Arrasvuori 2010) or evocative imagery (Hornecker 2010; Lucero and Arrasvuori 2010; Friedman and Hendry 2012).

Cards can provide context or instantiate knowledge (Bekker and Antle 2011; Friedman and Hendry 2012; Luger et al. 2015; Mora, Gianni, and Divitini 2017) and give concrete examples (Deng, Antle, and Neustaedter 2014; Mora, Gianni, and Divitini 2017) or instructions for activities (Friedman and Hendry 2012), however, establishing the right amount of information to have on a card is crucial—too much information can overwhelm the user and be time-consuming to read (Wetzel, Rodden, and Benford 2016), potentially distracting them from the flow of the activity that the tool is supposed to support (Deng, Antle, and Neustaedter 2014). Any descriptions and

definitions should therefore be succinct and easy to digest (Lockton et al. 2009; Lucero and Arrasvuori 2010).

As information should be accessible to those who need to digest it, the knowledge level of prospective users should be taken into account (Deng, Antle, and Neustaedter 2014). Unfamiliar content can pose problems (Lucero and Arrasvuori 2010) and lead to users ignoring cards or information that might help them in their tasks (Deng, Antle, and Neustaedter 2014; Mora, Gianni, and Divitini 2017) or them focusing more on interpreting a card than on the activity it is supposed to support (Mueller et al. 2014). Ideally, information should be written in simple, everyday language, avoiding wording that could be difficult to understand, including jargon (Lockton et al. 2009; Bekker and Antle 2011; Deng, Antle, and Neustaedter 2014; Mueller et al. 2014). However, in the simplification of complex concepts, detail is inevitably lost (Deng, Antle, and Neustaedter 2014)—trade-offs are often inevitable.

Visual appearance

Visual design can be used to reinforce information architecture and improve searchability (Deng, Antle, and Neustaedter 2014), using spatial layout, colour, iconography and typography, to make it easy to find specific cards, categories or types of content (Bekker and Antle 2011). It should be easy to differentiate cards (and categories) from one another (Deng, Antle, and Neustaedter 2014), and if a card is double-sided, the two sides should be visually distinct (Bekker and Antle 2011). Care should be taken with graphical imagery—some can be confusing or open to misinterpretation (Hornecker 2010; Lucero and Arrasvuori 2010; Deng, Antle, and Neustaedter 2014).

Structure

An effective information architecture will aid users in navigating a set of cards and finding the information they need, whether within a set or on a card. It should support easy scanning of the card deck (or subsets of multiple cards) and of the information types upon a card (Deng, Antle, and Neustaedter 2014). Categories have the potential to subdivide a deck of cards into meaningful units, but should be simple (Lockton et al. 2009) and easy to understand. One way to help physically differentiate or separate different card types and categories is through the use of tabs (Deng, Antle, and Neustaedter 2014).

Method of use

Any rules for card usage should again support the purpose of the cards and reinforce target behaviours (Lockton et al. 2009). A structured method may help (Lucero and Arrasvuori 2010; Mora, Gianni, and Divitini 2017) but different methods for different contexts (Lockton et al. 2009; Lucero and Arrasvuori 2010; Bekker and Antle 2011) or levels of experience (Wetzel, Rodden, and Benford 2016) may be appropriate—some users may want more guidance than others (Mueller et al. 2014). The learning curve for any method should not be too steep (Lockton et al. 2009).

5.3 Initial prototype

An initial low-fidelity prototype of the tool was created (Hanington and Martin 2012, 138), as an input to the next phase of design, instantiating some of the support suggestions derived from the empirical work in the previous chapter, as well as design considerations from the literature.

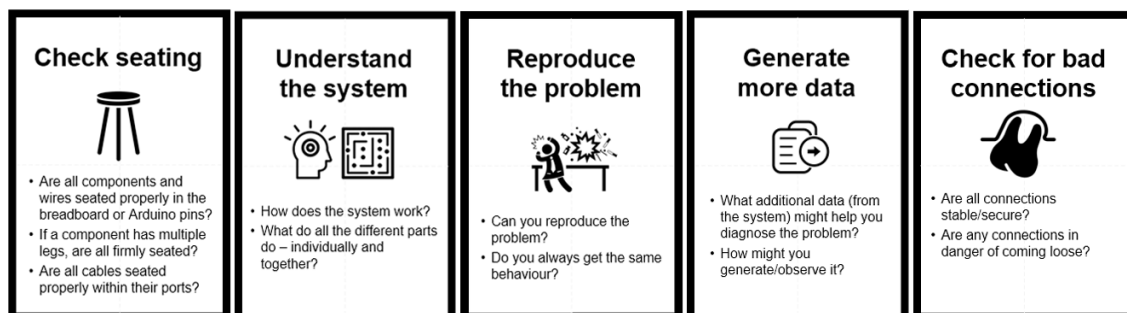


Figure 43. Content from the initial prototype of the troubleshooting support tool

To populate the card deck I identified forty candidate tactics for inclusion. This list of tactics was informed by both the literature and my empirical work, as follows.

- **Literature:** When developing the tactics coding scheme (section 4.2.6), I undertook a review of literature on debugging, troubleshooting and problem solving, looking for potential strategies or tactics that might be relevant to, or could be adapted for, a physical computing development context, or which might inspire specific patterns to look for within the data. Sources included not only academic literature, for example, previous work on novice, end-user and professional programming strategies and systems troubleshooting, but also non-academic sources, for example, Arduino books, debugging books aimed at professional software developers, electronics

troubleshooting guides and handbooks for creative problem solving. I looked for common themes within these sources, but also for domain and activity-specific methods and techniques. I returned to this research when developing the initial prototype of the card deck.

- **Empirical work:** Observing participants constructing a physical computing device had provided insight into problems that an end-user developer might encounter. Observing their troubleshooting behaviours had given me insight into tactics that proved useful in diagnosis, but also tactics that were problematic or less useful. Additionally, the recommendations for supporting end-user developers derived from my empirical work provided ideas for general and specific approaches that might help end-user developers, particularly in diagnosis.

I also returned to the list of circuit bugs experienced by participants. For each of these, I considered what tactics led to successful localisation, but also considered what else might have been helpful, taking into account the characteristics of these bugs, for example, the directness of feedback and how visible they were to the eye. This included tactics I had observed, but also additional ideas from the literature.

In a paired, open card-sorting exercise (Morville and Rosenfeld 2007, 255) with one of my PhD supervisors—a physical computing expert—the forty candidate tactics were grouped into eight initial categories.

Table 13 lists these categories, with a description of what they contained, some examples of tactics in each category, and finally, the support recommendation(s) from study 1B (section 4.4.2) that a category related to or instantiated. The full draft list of candidate tactics is available as Appendix I.

In addition, the deck also held four cards containing troubleshooting ‘wisdom,’ and three cards containing basic component wiring information: LEDs, TMP36 temperature sensor, and Arduino Uno board.

Table 13. Structure and content of the initial prototype card deck, and the relationship of the tactic categories to the support recommendations from study 1B

Category	In brief	Example tactics	Related recommendation(s) from study 1B
Understand (define) the problem	Identify and define the symptoms of failure, by using focused testing and analysing the results	Reproduce the problem, Check for abnormality	Recognise and define failure; Focused analysis of runtime behaviour / output Focused testing
Understand the system	Familiarisation with a system or its requirements, in order to understand it—to help diagnose or evaluate	Identify / trace dependencies; Check the brief	Recognise and define failure (understand what failure <i>doesn't</i> look like)
Inspect for build errors / faults	Visual checks, including for common or typical errors	Check for bad connections; Cross-check	Thorough inspection
Generate more data	Generate additional data that may be useful in diagnosis	Measure something; Logging statements	Focused analysis of runtime behaviour / output; Focused testing
Perform a test	<i>Planned</i> intervention or modification in order to test a hypothesis, establish boundaries or evaluate correctness	Swap working and non-working; Change test input	Focused testing
Try a quick fix	Common fixes that are quick to perform, ideally low risk and often easily reversed	Reverse orientation; Reseat	Focused testing; Also inspired by the concept of 'good' tinkering
Check component wiring	Access useful information about equipment	Copy an example; Get more help	Thorough inspection
Simplify	Make the problem space smaller in some way	Reduce dependencies; Divide & conquer	Problem decomposition

5.3.1 Proof of concept (informal pilot)

These cards were trialled in an informal, formative pilot / proof of concept study with two moderately experienced end-user developers (according to their own self-ratings), who develop physical computing prototypes to conduct their PhD research. In a ninety-minute session, I observed each use the card deck to troubleshoot buggy Arduino prototypes based on the simplified Love-O-Meter, while thinking aloud. The main findings from this informal evaluation were that 1) the deck contained too many categories and these could be more effective, 2) component information was seen as very useful, 3) some content could be improved, including

the tactic and category titles, and the questions that were designed to act as the main source of support, and 4) a better way to present and access the cards within their categories was needed, including a more distinct design for the category cards.

Additionally, having observed these two moderately-experienced end-user developers using the cards, I felt that to get the most out of the cards and the final (i.e. evaluation) study (Chapter 6), it would be better to focus more specifically on *novice*—i.e., less-experienced—end-user developers, rather than end-user developers in general, as it is reasonable to assume that less-experienced end-user developers would also have *less troubleshooting experience*. This would not preclude end-user developers with more experience from using the cards, rather it meant that design and content could be more tightly focused on the needs, preferences and experiences of novices, and evaluation could assess how this sub-population of end-user developers would fare with them.

Finally, an observation in respect to studying end-user developers' use of the cards, was that when using these cards for the first time, future participants would benefit from first having dedicated time in which to familiarise themselves with the deck as a whole, and the cards themselves, before undertaking any practical tasks with them.

In the next stage of the support tool's development, design input was sought from novice end-user developers, this now being the target population for the design and evaluation of the cards.

5.4 A study to inform the design of the card deck

Building further on my review of the literature, and informed by the pilot studies, I conducted a small study, to help further design the card deck, through input from representative users.

I ran two focus group sessions, each involving a pair of novice end-user developers (ages ranging from 30 to 42; one female pair, one male-female pair). All were new to Arduino and had limited experience of both programming and electronics. Potential participants were sent a study information sheet prior to taking part (Appendix K) and full ethical clearance to undertake the study was granted by the university (Appendix J).

In a preparatory session with one of my supervisors (physical computing development expert), the forty candidate troubleshooting tactics were discussed, one by one, and consolidated where we agreed there to be redundancy or duplication. Together we whittled the set down to thirty-four tactics, tentatively grouped into seven categories through another paired card sort that used the existing categories as a starting point. Based on this set, materials were developed for use in the focus group sessions.

During the focus group sessions, participants were asked for feedback in terms of card information content, physical form, and visual appearance, as well as categorisation of the cards. With participants' permission, I video-recorded the focus groups, and took notes during them. The notes and transcripts of the video recordings were used for analysis.

5.4.1 Focus group sessions

Participants first completed a background questionnaire capturing their experience in physical computing and were interviewed, as a pair, about their experience of Arduino and the environments where they typically used—or would like to use—Arduino, priming them for considering a context of use for the cards. Participants then undertook four exercises. Pair discussion within each exercise was guided by a prompt card, suggesting things they might wish to consider, for example, *“Are there any obvious advantages/disadvantages to [...]”*.

5.4.1.1 Exercise 1: Physical card format

In the first exercise, the pair of participants considered four physical card formats—a smaller, playing card size and one double that size, in both landscape and portrait orientations—and ranked these in order of preference. Cards were blank and cut from a decent cardstock, so that participants would focus on physical form and handling, rather than potential content or visual design, and could experience what a deck might be like to handle—multiple cards of each option were provided. An Arduino prototype (the simplified Love-O-Meter) was placed in front of each participant, as it might be during an actual troubleshooting task.

Outcome: Participants much preferred the smaller cards, in portrait orientation, being a familiar, standard playing card size and easier to handle than the larger cards:

“All card games are this size. There is a very good reason for it. They feel very nice in the hand, and you can flick through them very easily. (PB1)”

Participants thought smaller cards would take up less space and be laid out easier when there was not much room, particularly if they wanted to work with several cards at once. They felt landscape cards in both sizes to be harder to hold and flip through.

5.4.1.2 Exercise 2: Information content

Prior to the sessions, I created sample information content for two tactics: one a lower-level tactic (*Inspect for poor connections*), intended to prompt an end-user developer to visually inspect their circuit for a particular type of bug; the second, a higher-level tactic (*Isolate part of the system*), requiring an end-user developer to think about how they might simplify and test their prototype to narrow in the location of a bug. Each piece of content was printed on blank paper. In the second exercise, participants were asked to consider three different types of sample information content for each of the two tactics, and rank these by preference:

- ‘*Questions to ask*’—Designed to encourage thought or reflection. For example: “How could this help you to narrow in on the cause of failure, or rule something out?”.
- ‘*Can apply to*’—Information to guide troubleshooting to the bug location. For example: “Jump wire ends in breadboard holes or Arduino pins”.
- ‘*Ways to apply*’—Things to do, supporting specific trouble-shooting activities. For example: “Check component legs in the breadboard”.

Outcome: ‘*Questions to ask*’ was ranked as most useful, followed by ‘*Ways to apply*’. Encouraging reflection through the use of questions was appreciated by most participants:

"I think 'Questions to ask' should be the first thing. Because that's how you diagnose a problem... you'd start from there, right?" (PA2).

"this is what I would love to have, to spark some thinking in myself, so I could kind of direct my investigations" (PB2).

However, an exchange in Pair A shows that even if novices see value in thinking through things themselves, some may prefer to be told what to do. As one of the aims of the support tool is to encourage novices to be more thoughtful troubleshooters, this response is worth noting.

PA1: "I just want 'ways to apply', and I'm like (mimes following the 'Ways to apply' list)... but I guess I'm not thinking about what I'm doing".

PA2: "But you don't know what's the problem yet, which is why you ask the questions, that's the way you decide."

PA1: "I don't like it, but I think you're right"

PA2: "You don't want to ask questions?"

PA1: "No, I want it to tell me 'Do this, do this' (laughs)"

I also observed that having one main or 'lead' question may lead to it being treated as a binary determinant of usefulness, rather than prompting thought:

"If the answer [to the lead question] is 'No', then you can ignore it (the card)" (PA2).

What I took from this is that a short summary of the tactic, using active wording, might be better than a main/lead question, in helping novices decide whether a card is useful. It may also be helpful in explaining *why* or *how* a particular tactic can be useful.

While participant did like the concrete instructions of '*Ways to apply*', they appeared to assume that both '*Ways to apply*' and '*Can apply to*' would act as comprehensive checklists, which is unfeasible—a card could not contain instructions for every possible context in which a particular tactic might be applied. There is therefore a risk that '*Ways to apply*' or '*Can apply to*' may lead novices to assume these are the only things they need to do/check. Equally, care should be taken that questions are always worded to prompt thinking, for example, '*Where could there be a poor connection?*' rather than as potential things to check off, e.g. '*Are all components seated securely in the breadboard?*'

5.4.1.3 Exercise 3: Visual design and content

In the third exercise, participants then considered 30 different designs, each for a potential front or back of a card. As well as in size, orientation and information, designs differed in colour coding, typography, titling (full titles and single-word titles), and iconography (including size and location). Through discussion, each pair created three potential 'whole card' designs, ranked by preference. There were no restrictions, for example, cards could be single or double-sided, and different orientations could be chosen for the front and back of a single card. Participants could also sketch alternative card designs, should they wish to do so—blank card was provided for this purpose.

Outcome: All participants felt that, as novices, having iconography, categorisation and colour coding would aid understanding, recognition, and selection, and that single-word titles were too ambiguous. The top-ranked card created by each pair was identical: a smaller-sized, portrait-oriented, double-sided card, with a distinct, uncluttered front (full title, large icon, brief summary) and more detailed information on the rear.

5.4.1.4 Exercise 4: Categorisation (card deck structure)

Finally, in the fourth exercise, participants performed a card sorting exercise, using the set of 34 tactic titles and seven category titles, in order to inform the information architecture of the card deck. This was a closed card sort, as it was felt that having prompts for category names would be more likely to lead to some consensus across the two groups, than allowing novices to create categories for tactics with which they may not be familiar, however, participants were free to suggest new category names, or changes to existing ones. Working through the set, participants discussed each tactic and, as a pair, agreed on which category to put it into. If unsure, they could also place tactics into a “?” category.

Outcome: While both pairs sorted most tactics into the categories to which I had originally assigned them (Pair A 26/34; Pair B 20/34), this exercise helped to identify some confusing or ambiguous wording and the need for some categorisation changes.

5.5 The Tactical Troubleshooting toolkit

Informed by the findings of the focus groups, the card set was revised. Twelve tactics and two categories were renamed or reworded slightly, to make them easier to understand (for example, changing ‘Check the type’ to ‘Check the type(s) used’, as focus group participants had misinterpreted the former as ‘check the (typed) code’). Five tactics were also reassigned to different categories and three categories removed. Further discussion led to a new category of tactic: ‘*Stop... think*’, explicitly encouraging thinking/reflection, and the addition of two new tactics.

As well as tactics, the card set also included two other types of card: *Best Practice cards* (previously referred to as *troubleshooting wisdom*), in their own category, and *Component cards*, assigned to the *Get Help* category, as they contain information about specific components—these latter cards were not used in the final study of this thesis (Chapter 6). The list of cards used in the final study is available as Appendix L.

The troubleshooting card set currently comprises 46 cards: 36 Tactics cards and four Component cards in five categories, as well as six Best Practice cards. (see Table 14, Figure 51;

Figure 52). Additionally, each category has what is essentially a header or index card, which I refer to as *Category cards* (e.g., Figure 46). There are therefore 52 cards in total.

Table 14. Card categories and their contents

Category	Contents	Description
Analyse runtime behaviour / data	1 x Category card 8 x Tactic cards	Tactics encouraging end-user developers to define symptoms of failure, and analyse runtime behaviour or output in a targeted way, including through the use of specific tools.
Inspect hardware / software	1 x Category card 14 x Tactic cards	Tactics to make end-user developers aware of specific types of inspection they can perform—what they can look at and why. Most are essentially hypotheses for where faults may lie, including common problems, instantiated in the form of tactics.
Conduct a test	1 x Category card 7 x Tactic cards	Tactics encouraging end-user developers to perform specific tests which may help them to diagnose their problems, localise faults, and evaluate behaviour, as well as approaches to help them narrow in more closely on areas of functionality that may contain faults. All of these tactics involve making changes of some kind.
Stop... think	1 x Category card 4 x Tactic cards	Tactics encouraging end-user developers to take a step back from <i>action</i> , to <i>think</i> or <i>reflect</i> instead
Get help	1 x Category card 3 x Tactic cards 4 x Component cards	Tactics suggesting ways in which end-user developers might use external help, as well as cards containing information about common components and how to use them—currently these only include components used in the Love-O-Meter prototype.
Best practice	1 x Category card 6 x Best Practice cards	Each card instantiates a piece of good practice that end-user developers should bear in mind when troubleshooting.

Figure. 44 shows the front of a card from each category. Images of the full set of tactics and best practice cards (front and back) can be viewed as Appendix M. The cards will now be described in greater detail.

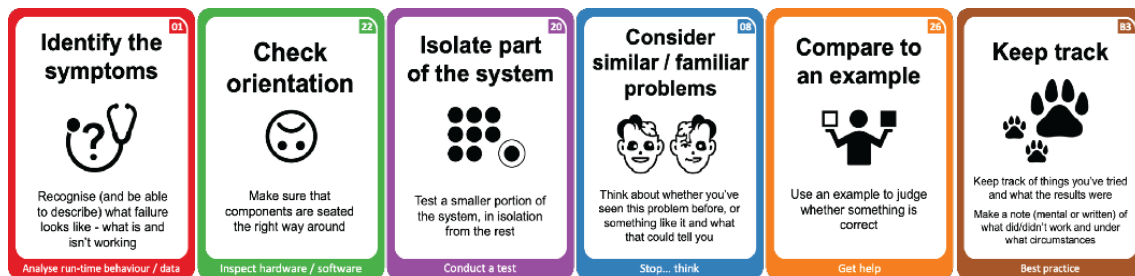


Figure. 44. An example card from each category

5.5.1 Tactic cards

Physical form: The tactic cards (Figure 45) are rectangular, and of standard playing card dimensions (64mm x 89mm)—a familiar size that is easy to hold and manipulate, including when holding multiple cards. Cards are cut from a decent-quality card stock that allows easy shuffling through multiple cards—several types of card stock were physically tested in order to establish this—and card corners are rounded, for easy handling (section 5.6 discusses the production process in greater detail). Cards are double-sided, and content on both sides is portrait-oriented, making it easy to simply turn over a card and read the content without further reorientation.

Information content: The cards aim to scaffold end-user developers’ troubleshooting of problems—particularly circuit bugs—in physical computing prototypes, and therefore, contain content useful to this process—*tactics* which can be used to diagnose and fix bugs, and evaluate the success of any fix attempts. As described in section 5.3, the choice of tactics was informed by the empirical studies described in Chapter 3 and Chapter 4, as well as academic and non-academic literature on novice, end-user and professional debugging, physical computing, systems troubleshooting, electronics troubleshooting, and general and creative problem solving.

As different types of approaches can be useful in troubleshooting, including both system-specific procedures and common problem-solving techniques (Gick 1986), the tactics in the card deck deliberately vary in their specificity—some, for example the ‘*Serial Monitor*’ card, are *specific* to some aspect of circuit bugs, physical computing or the Arduino platform, while others are more *general*, for example problem-solving tactics such as ‘*Divide and Conquer*’. Also, while some tactics are very *practical*, including tactics for localising common circuit bugs, such as ‘*Check orientation*’, others, such as ‘*Question Your Assumptions*’, are more *thought-provoking* or *vague*. Where appropriate, tactics are also worded in such a way that they might be adapted to different contexts, for example, ‘*Check Values*’ could apply to the size (in ohms) of resistors used in a circuit, variable values within a program, or the values of settings configured within the IDE. Therefore, much of the content in the card deck is relevant for troubleshooting physical computing problems in general, not just circuit bugs.



Figure 45. Tactics card design, front (left) and rear (right)

Both sides of the tactics cards contain information—adopting a *layering* approach (Shneiderman 2003), with simple information on the front—title, icon and simple description, i.e., enough to give a novice end-user developer some indication of what the tactic is about—and more detailed information on the rear.

At the top of the rear of the card is a short explanation of why or how a tactic might be useful, providing context for its adoption. Beneath this, rather than a list of instructions, there is a bulleted list of questions, to guide end-user developers in using the tactic. These questions are deliberately worded to prompt thinking or reflection (Morgan and Saxton 1991, 63), reinforced by the title of this panel, i.e., '*Think about...*', and are similar to the prompts used to help participants overcome their difficulties, in Perkins and Martins' study of novice programmers' knowledge and strategies (Perkins and Martin 1986).

Wording of all titles, descriptions/explanations and questions uses simple, everyday language, avoiding jargon where possible, although in some cases, terms common to physical computing have been used—for example, '*pinout*'—to enable end-user developers to use these terms in searches for external help—where domain-specific terms have been used, the description, written in simple language clarifies their meaning. Titles and descriptions are succinct, so as not to overwhelm end-user developers with too much information, or take up too much space on the card.

Visual design: Visually, the cards have a simple but striking and eye-catching design, making good use of white space and different sizes and weights of typography to draw attention to—and differentiate between—areas of content. The two sides are visually distinct, making it easy to determine, at a glance, which side is currently visible.

The front of the card contains three main elements: the title, in a large, bold font, a black and white icon, and a short explanation of the tactic, in the form of a simple summary of what using the tactic will entail. I chose black and white icons from the Noun Project icon repository ('Noun Project' n.d.)—simple rather than complex imagery—with each relating to the title of the card in some way, in some cases playing upon the wording, to aid memorability, for example, an image of a wooden log on the '*Logging (Print) Statements*' tactic. These icons serve as visual interest, breaking the monotony of text, but also act as secondary information that can help an end-user developer to interpret the meaning of the card, and aid in recognition, including when an end-user developer is scanning or shuffling through several cards, or when a card is turned over—a smaller version of the icon is repeated on the rear of the card.

Colour-coding is used to differentiate between different categories of cards, using a colour scheme designed for maximum visual difference (Harrower and Brewer 2003)—each card is bordered in the colour of the category to which it belongs, and bands of the same colour are used to separate the different panels of content on the rear. Additionally, the tactic title is repeated in a small, unobtrusive font size at the top of the rear of the card, and the category name appears at the bottom of both sides.

Structure: Informed by the focus groups with novice end-user developers, the tactics are organised in five categories: '*Inspect Hardware/Software*', '*Analyse Runtime Behaviour/Data*', '*Conduct a Test*', '*Get Help*' and '*Stop... Think*'. The number of tactic categories was deliberately kept small, so not to overwhelm novice end-user developers with too many options. Category titles are worded to indicate the main type of activity involved in the cards they contain, for example, '*Inspect...*' implies visual inspection, while the '*Conduct a Test*' category deliberately classifies some of the *good* speculative changes observed in the previous empirical studies as 'tests' to reinforce a more thoughtful, *hypothesis testing*-focused approach to these kinds of changes.

5.5.2 Category cards

The category cards act as header or index cards (Figure 46) for the categories of tactics cards and the Best Practice cards category. As described in the previous section, colour is used to visually differentiate the different categories, and the category cards are slightly taller than the other cards, so that the titles are visible above them, facilitating visual scanning of the category names, and easy selection of cards in a particular category. On the rear of each category card is a bulleted list of the cards that the category contains, making it easy to visually scan the content of each category, including when the cards are placed in a stand (Figure 50).



Figure 46. Category card design, front (left) and rear (right)



Figure 47. Best Practice card design

5.5.3 Best Practice cards

A sixth category of cards—‘Best Practice’—contains cards which suggest best/good practice that should be followed when troubleshooting, for example, ‘Keep track’, which encourages end-user developers to keep track of their troubleshooting actions and the results of these efforts. These cards (Figure 47) follow a similar design to the tactic cards, but are currently single-sided and contain a description rather than questions, as the aim is to inform, rather than to prompt thinking something through.

5.5.4 Component cards

The component cards (Figure 48) provide information about specific components—currently these are components used in the Love-O-Meter prototype: Arduino board, LED, resistor and TMP36 temperature sensor. These cards are larger (89mm x 128mm), as they contain far more information and more complex imagery, including component pinout images, specifying correct connection types, and basic circuit wiring information, as well as other information key to using or controlling the component. These cards were not used in the final study (Chapter 6), as the focus of that study was on use of the tactics cards (see section 6.2.4.5), my main interest being in supporting the troubleshooting *process* (see section 7.3).

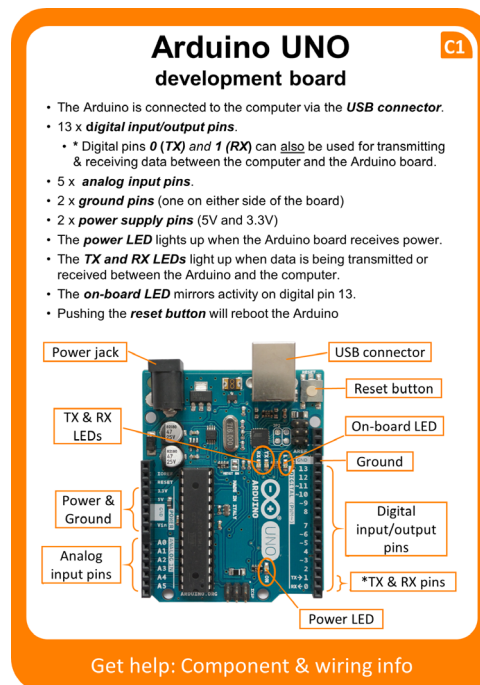


Figure 48. Component card design

5.5.5 Playmat

The findings from the analysis reported in Chapter 4 suggested that end-user developers might benefit from support in structuring and planning their troubleshooting. Inspired by boardgames and some card games, I designed a *playmat* (Figure 49) that novice end-user developers can use to help structure and support the process of troubleshooting.

A playmat is, generally, a portable surface upon which cards and/or other relevant items can be placed during a game. Playmats may be plain, or purely decorative, but for some games playmats contain designated areas—zones—in which cards can be placed, whether for organisation/storage, or to support and/or guide the method or rules of play.

The playmat designed for the troubleshooting toolkit gently guides end-user developers towards using the cards in a systematic way. It does this by providing two specific, demarcated areas/zones in which cards can be placed: 1) a ‘shortlist’ area (*Ideas*), to hold a selection of potential tactics to try—thus encouraging end-user developers to plan their actions, considering and prioritising different options/hypotheses—and 2) an ‘active’ area (*Current*), to hold the current card(s), that is, the chosen line of enquiry. Each of these zones is just slightly larger in dimensions than the tactics cards, visually implying that cards should be placed on them.

The playmat additionally serves to remind end-user developers of good troubleshooting process: a flowchart in the centre of the playmat reinforces the cycle of ‘Diagnose → Fix → Evaluate’, encouraging end-user developers to diagnose before attempting fixes, and then to evaluate the result of any fix attempt before any further action.

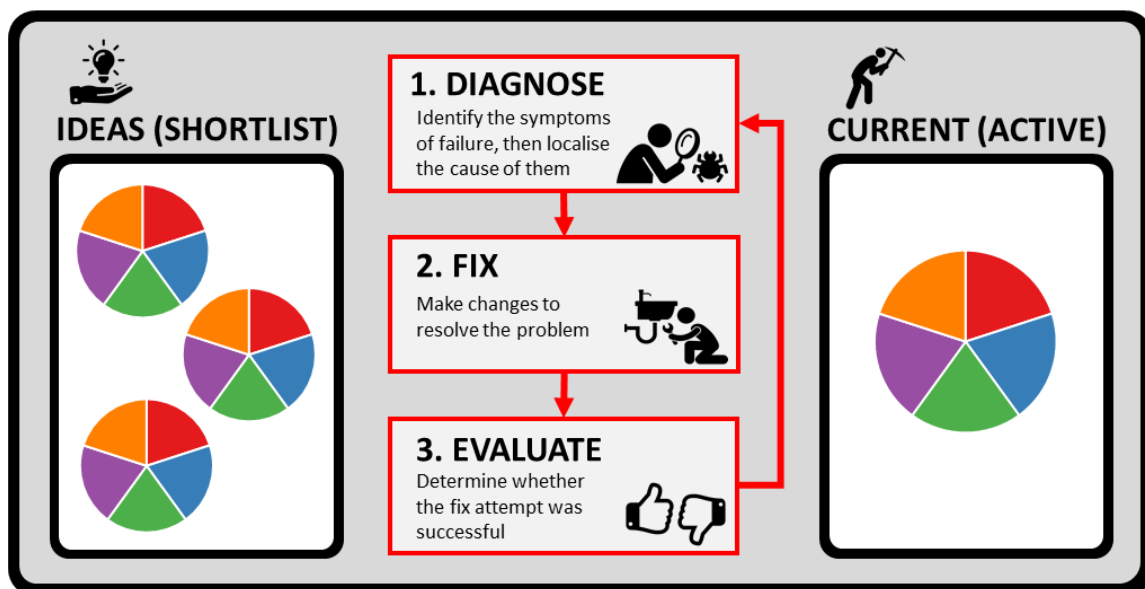


Figure 49. Troubleshooting toolkit playmat

5.5.6 Card stand

Finally, to hold and display the cards in a structured, space-efficient way, I 3D modelled and 3D printed a three-tier stand (Figure 50). When cards are placed in the stand, within their categories, it provides end-user developers with visual prompts about the different tactical approaches available to them and makes the cards in a category easier to access. There is an additional, wider slot on the third tier, to hold the larger component cards.



Figure 50. Card stand

5.6 Card production process

In parallel with the design research, the production of the cards evolved, through experimentation with different materials and methods. While conscious that I was developing a prototype, I also wanted to ensure that the production quality of the cards would not negatively affect any use or evaluation of the tool as a potential support medium in the final study, and that it would afford the types of handling that are associated with card use, for example, holding, fanning, dealing, picking up, stacking and shuffling.

The earliest prototypes were simply printed on A4 printer paper and cut to size with a rotary guillotine, but these 'cards' were too flimsy to handle effectively. Heavier weight paper (light card) was better, but still had too much friction, for example, when shuffled or fanned. Clear plastic protective card sleeves also improved sturdiness while retaining flexibility, however slipperiness of the sleeves, even matte versions, affected both shuffling and stacking into piles. I

sourced several different weights and textures of card stock, cutting and testing the handling of multiple blank cards. A linen-textured card stock, similar to that used in some professionally manufactured playing card decks, proved optimal.

Card designs were created in Microsoft PowerPoint—the use of Slide Masters to compose design templates made it easy to add new cards, or rapidly change the design of all/multiple cards at once. I later experimented with the use of Adobe Illustrator, as a more professional design tool, but PowerPoint proved simpler and more efficient for my workflow. This also meant that the designs could be easily adapted by anyone with access to commonly available software and basic presentation editing skills.

When, as a result of the design focus groups (section 5.4), the card design became double-sided, this presented further challenges. Reliably accurate print registration proved too difficult to achieve when printing double-sided with a home inkjet printer. Better results were achieved by printing both sides single-sided—Appendix M shows these designs at a smaller scale—and then folding the cards, which meant sourcing a lower weight of linen-textured card stock, to avoid the cards being too thick. Rounding corners with a corner cutter tool, also for better handling, proved laborious with a large number of cards, as was even cutting the straight edges with a guillotine. I therefore I designed a round-cornered cutting template for use with a Cricut Maker die cutting machine.

For the support toolkit used in the evaluation study (see section 6.2.4.5), the card designs were printed onto lightweight linen textured card stock and a Cricut Maker machine used to automatically cut out multiple cards at once. Card designs were then folded in half and spray glue used to join the two sides. Finally, both faces of the cards were sprayed with a very thin coat of clear resin, one which did not affect shuffling or print quality, to protect the printed design and prevent smudging (for example, due to handling) that might affect text legibility.

5.7 Discussion

In this chapter I have described the design, development and production of a novel, physical card-based tool to support novice end-user developers in troubleshooting physical computing problems during development, particularly those related to circuit bugs.

The general aim of the card deck is to provide novice end-user developers with a wide range of troubleshooting tactics that can be used to improve diagnosis of physical computing problems, fixing of bugs, and evaluation of fixes, and to facilitate thinking/reflection during this process. The goal is not to give exhaustive and prescriptive checklists of instructions, but rather to encourage a creative and exploratory approach to troubleshooting.

The troubleshooting card tool was inspired by popular creativity-support card decks, and content was informed by empirical work to identify the problems that end-user developers encounter when developing a physical computing device, and analysis of their natural troubleshooting behaviours, as well as a review of the academic and non-academic literature on software debugging, hardware troubleshooting and general problem solving. The design of the card deck was informed by a design review of the literature on card-based tools, identifying key considerations when designing card-based tools, and focus groups with novice end-user developers.

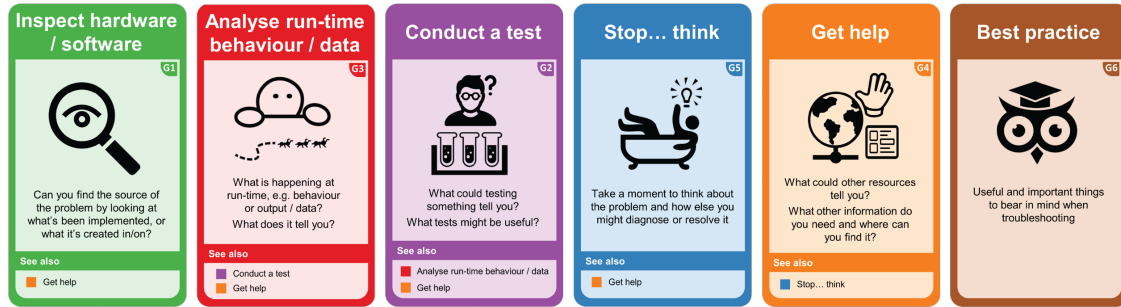
The next chapter describes an evaluation of these cards in a study with novice end-user developers. In this study, I investigate the role that these cards might play in a troubleshooting process and any effect on the outcomes thereof, as well as how support in this medium might be perceived by novice end-user developers.

A. Tactic cards (front sides only)

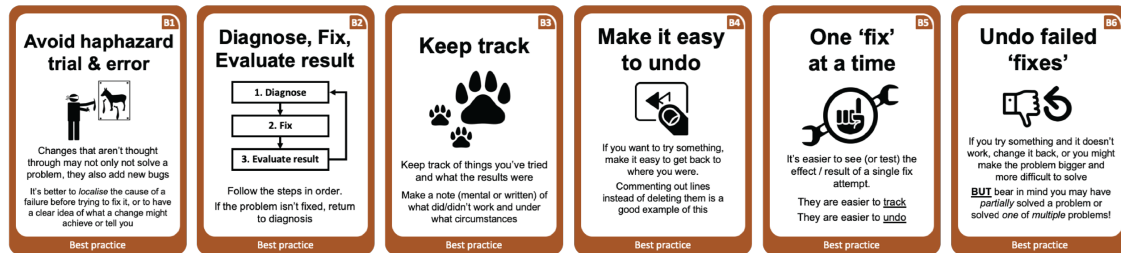


Figure 51. Tactics cards (front sides only)

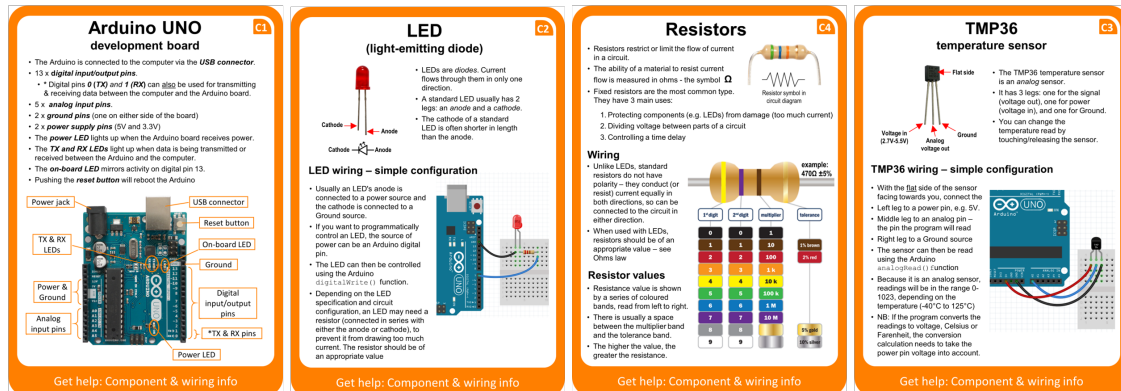
B. Category cards



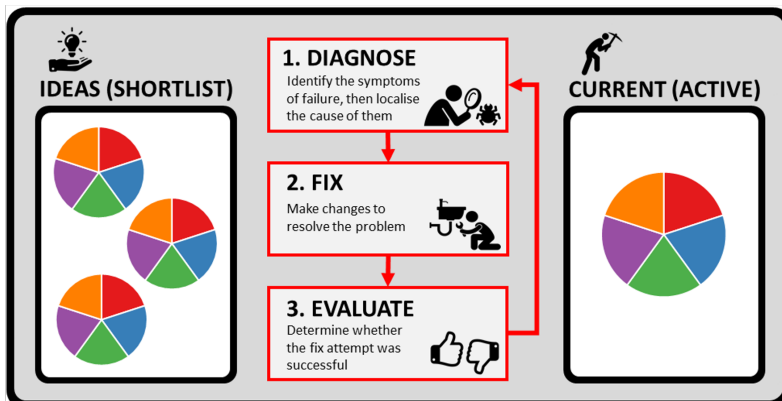
C. Best practice cards



D. Component information cards



E. Playmat



F. Cards stand



Figure 52. Category cards (fronts only), Best Practice cards, Component cards, Playmat, Card stand

Chapter 6

Evaluating the troubleshooting support cards with novice end-user developers (Study 2)

6.1 Introduction

This chapter describes and reports an evaluation of the physical card-based support tool (Chapter 5), in a study with novice end-user developers. The study addressed the following thesis-level research question:

TRQ4 What role might a card-based tool play in supporting end-user developers in the process of troubleshooting circuit bugs in a physical computing prototype?

I broke this down into two study-level research questions:

RQ1: What effect does a physical card-based support tool have on end-user developers' success in troubleshooting circuit bugs in physical computing prototypes?

RQ2: How do end-user developers view the physical card-based support tool, having used it to troubleshoot circuit bugs in physical computing prototypes?

6.2 Method

6.2.1 Overview

To answer the research questions, I conducted an empirical, within subjects user study in which twenty novice end-user developers—Arduino users—each undertook two hands-on troubleshooting tasks—one with and one without the support tool—while thinking aloud.

In each task, participants had a set amount of time to locate and fix preseeded circuit bugs in an Arduino prototype—each a prebuilt instantiation of the simplified Love-O-Meter device used in my first study—until its behaviour met a given specification.

I collected a range of data for this study, including video recordings of the tasks, as well as a questionnaire and interview capturing participants' opinions of the support tool.

While the results report a comparison of performance measures such as task success and bug fixing success, analysis focuses, for the most part, on the qualitative analysis of participants' subjective feedback about the support materials.

6.2.2 Study design

The main goal of the study was to evaluate the card-based support tool. I therefore chose to conduct the evaluation as a user study or usability study—*"Representative users attempting representative tasks in representative environments, on early prototypes of computer interfaces"* (Lazar, Feng, and Hochheiser 2017; citing Lewis 2006)—in which the support tool—the prototypical interface in this instance—was assessed through observation of hands-on use and the analysis of feedback about the experience of using it.

I considered—and rejected—several alternative methods. For example, as I wanted to directly observe participants using the tool, rather than *only* relying on subjective feedback, I decided against a diary study. I also considered whether to observe end-user developers troubleshooting in pairs, but rejected this as adding complication to recruitment that I was already concerned about (section 7.2 discusses some of the challenges I encountered when recruiting for my studies).

A Within Subjects study (Lazar, Feng, and Hochheiser 2017, 49) in which each participant undertook tasks with and without the support tool, would allow me to compare performance measures. It also meant that participants would have experienced troubleshooting with and without the support tool, and could consider this in their feedback.

Other factors played into my choice of a Within Subjects, rather than a Between Subjects study design. Recruiting my quota of twenty participants had not been easy in the first study. In a Within Subjects study, all participants would experience the support tool, providing the maximum number of data points in respect to the research questions; this might also be useful

for further refining the design of the tool. Secondly, a recognised challenge when studying development is that variability in skill/expertise of developers can affect results (Ko, LaToza, and Burnett 2015), for example, performance and user experience data. One way to control for this is by ensuring that participants recruited are equally skilled, however, besides my recruitment concerns, to my knowledge, no reliable measure of expertise in physical computing exists as yet, and it would have been impractical to design and validate one within the constraints of my PhD work. Also, the effect of different *experience*, for example, previous exposure to certain bugs, could also affect results. A Within Subjects design minimises the effect of individual differences.

One disadvantage, however, is the potential for a learning or transfer effect across conditions, which can also affect results. Counterbalancing—varying the order of exposure to conditions—is a typical way to mitigate this. In this study, I counterbalanced the order of exposure to the support tool—participants were randomly assigned to one of two groups, determining whether they had access to the support tool in the first or second task. However, as both task prototypes were based on the simplified Love-O-Meter device, a learning effect still seemed likely. Despite this, I chose to proceed with this study design, for reasons explained further in Section 6.2.4.5 (in *Buggy prototypes*), taking care to minimise the learning effect to the greatest degree possible, for example in the design of the two prototypes and the order of exposure to them.

6.2.3 Participants

Once ethical clearance had been granted by the university (Appendix N), twenty adult, novice end-user developers were recruited.

Table 15 shows the inclusion/exclusion criteria used to screen prospective participants.

Eligibility was narrower than in the first study, in respect to expertise—participants must have had at least *some* experience of using Arduino, but also had to consider themselves to be *novice* Arduino users. In addition, they also had to consider themselves to be novice in either programming or electronics, or both. As before, participants needed to be *end-user developers* in physical computing and should have been exposed to using LEDs *and* at least one type of analog sensor.

Table 15. Study 2 inclusion/exclusion criteria for participation

Inclusion criteria	Exclusion criteria
Adult: Aged 18 or older	Aged under 18
At least <i>some</i> practical (hands-on) experience of using the Arduino platform, with, as a minimum, <i>both</i> of the following (although not necessarily in the same project):	No practical (hands-on) experience of using the Arduino platform.
- Exposure to using LEDs in an Arduino project AND - Exposure to using at least one type of analog sensor in an Arduino project	Exposure to using <i>either</i> LEDs <i>or</i> analog sensors in an Arduino project, but not both.
End-user developer: Has only developed physical computing prototypes/devices for own use.	Previously or currently employed/commissioned specifically to develop physical computing prototypes/devices.
Novice in programming, electronics or both	Expert in programming <i>and</i> electronics
Able to attend a 1.5-hour session in-person	Unable to attend a 1.5-hour session in person

6.2.3.1 Recruitment

Participants were recruited via hackspaces and other Maker community groups, using flyers (Appendix O) and mailing lists, and through personal networks. As I wanted to recruit novices, I also targeted university programmes where physical computing was taught. The timing of the study meant that data gathering coincided with earlier weeks of first semester teaching, which was ideal for catching potential participants at exactly the right time in their studies—they would, by this stage, know enough to take part, but not enough to be considered *too* experienced. The call for participation was also posted on social media (Twitter and Facebook) at regular intervals, and others were asked to share these posts with their networks. A poster was also put up at several universities. Given the time commitment required (2 hours), I offered participants an incentive of a £20 Amazon gift voucher as a token of thanks for taking part.

All respondents were sent a copy of the participant information sheet (Appendix P) explaining the study and eligibility criteria in full, as well as what participation would entail, so that they could determine their eligibility and make an informed decision about whether they wished to participate. Respondents who met all of the criteria for participation were invited to attend an individual session, lasting two hours, in the Interaction Lab at City, University of London.

6.2.3.2 Who took part?

A background questionnaire (Appendix R, described further in section 6.2.4.2) was completed by participants at the start of the session they attended. It captured participant demographics, as well as data about experience, perceived expertise (self-rated) and training. Data from the completed paper questionnaires were entered into an Excel spreadsheet. Length of experience was converted to a decimal figure (years), and all data were summarised using descriptive statistics. Table 16 shows the age, gender and occupation of those who took part. See Appendix W for a table summarising the remainder of the data.

Twenty novice end-user developers took part in the study—10 male and 10 female, all adults, ranging in age from 20 to 51, with a mean age of 32.75 years (SD=10.84). I will now describe the sample in greater detail.

Table 16. Study 2 Participants

Ptc	Age	Gender	Occupation
P110	41	Male	Web Developer
P120	28	Female	PhD student (Media & Art Technology)
P130	26	Male	Masters student (Human-Computer Interaction)
P140	39	Male	Electrician
P150	21	Male	Undergraduate student (Biomedical Engineering)
P160	21	Male	Undergraduate student (Engineering)
P170	35	Female	Creative
P180	51	Female	Sound engineer
P190	21	Male	Undergraduate student (Computer Science)
P200	38	Male	Charity consultant
P210	31	Male	Lab technician
P220	48	Female	Masters student (Computational Art)
P230	20	Female	Undergraduate student (Mechanical Engineering)
P240	51	Female	Masters student (Computational Art)
P250	20	Female	Undergraduate student (Creative Computing)
P260	28	Male	PhD Student (Media & Art Technology)
P270	47	Female	Finance
P280	21	Female	Masters student (Design)
P290	37	Male	Freelance educator (Primary school)
P300	31	Female	Research Fellow (Human-Computer Interaction)

Occupation

Eleven participants were students (four Undergraduates, four Postgraduate Masters students and two PhD research students), while nine were primarily working/employed in some capacity.

No participant had ever been employed as an electronics engineer, although one (P140) stated their current occupation as ‘Electrician’—in the screening they had only mentioned having had *some* exposure to practical electronics and did not consider themselves to be an expert. Two engineering students (Undergraduate) took part, but both reported having little programming experience.

Four participants had been employed as programmers at some point—one was employed as a web developer at the time of the study, while two were Computer Science students (MSc and BSc, respectively), and one was employed as a HCI researcher, although without any programming responsibilities.

Training / instruction

Over half of the participants had received some sort of training or instruction in Arduino (14/20)—ten in the form of a university/HE institution module (previous or current), three only attended short workshops, and one only reported using online materials)—with slightly fewer (12/20) having had training/instruction in electronics. All participants, however, had received some form of training/instruction in programming and for sixteen this had been in the form of at least one module at university or another HE institution.

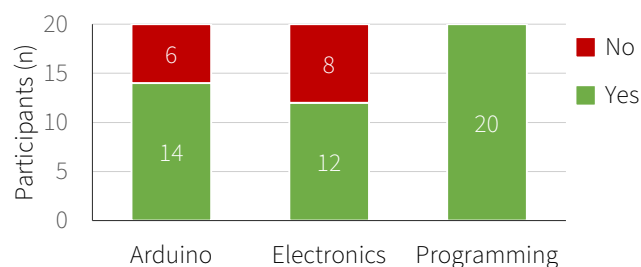


Figure 53. Participants' training

Length of experience

On average, participants had been using Arduino for less than half a year (Mean=0.47 years, SD=0.35), but had twice as much electronics experience (Mean=1.04 years; SD=0.65) and again more programming experience (Mean=2.54 years; SD=2.64). There was far more variance in participants' length of programming experience than in their electronics or Arduino experience.

Table 17. Study 2 Participant length of experience (in years)

Time spent (in years)	Mean	SD
Arduino	0.47	0.35
Electronics	1.04	0.65
Programming	2.54	2.64

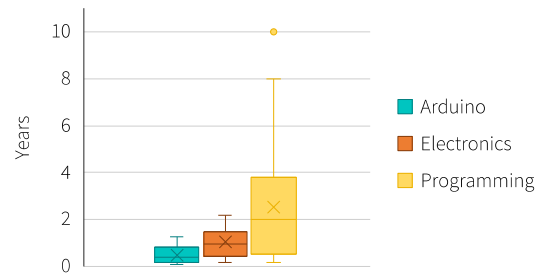


Figure 54. Participants' length of experience (in years)

Self-rated expertise

Participants also, on average, rated their programming expertise highest, their electronics expertise to be lower, and perceived themselves least skilled in the use of Arduino, although there was approximately twice as much variance in the electronics and programming expertise ratings than there was in respect to Arduino. Only two participants rated themselves as complete beginners (rating=1) in any of the three areas (electronics: P110 & P180) and no participant considered themselves to be a complete expert in any of the three skills.

Table 18. Study 2 Participant perceived expertise (1-7)

Perceived expertise	Mean	SD	Median
Arduino	2.70	0.73	3.00
Electronics	2.95	1.32	3.00
Programming	3.70	1.42	3.50

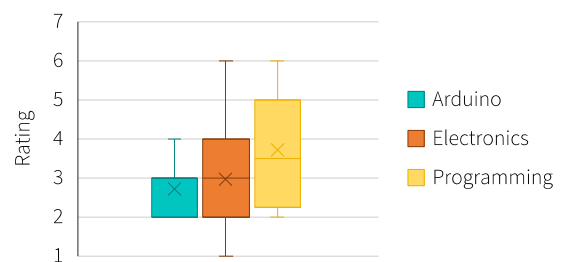


Figure 55. Participants' self-rated expertise

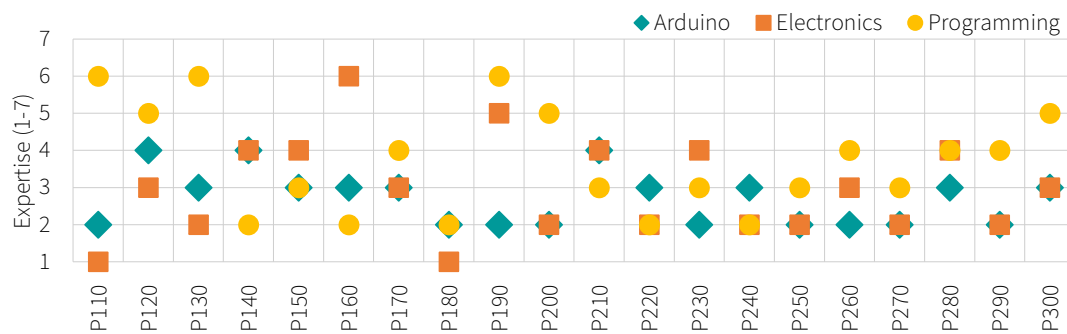


Figure 56. Individual participants' self-rated expertise, from 1 (Complete beginner) to 7 (Complete expert)

Self-rated troubleshooting expertise

Given their experience and expertise ratings, it is unsurprising that participants considered themselves, on average, to be most skilled at troubleshooting program bugs, although this time least skilled in electronics troubleshooting, rather in troubleshooting bugs in Arduino projects.

Table 19. Study 2 Participants perceived expertise in troubleshooting (1-7)

Perceived expertise	Mean	SD	Median
Arduino projects	2.65	0.88	2.50
Circuit bugs	2.50	1.50	2.00
Program bugs	3.35	1.53	3.00

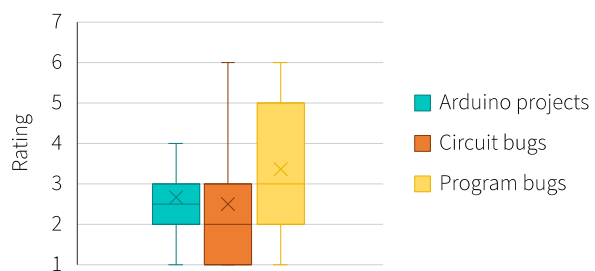


Figure 57. Participants' self-rated troubleshooting expertise in Arduino, Electronics, Programming

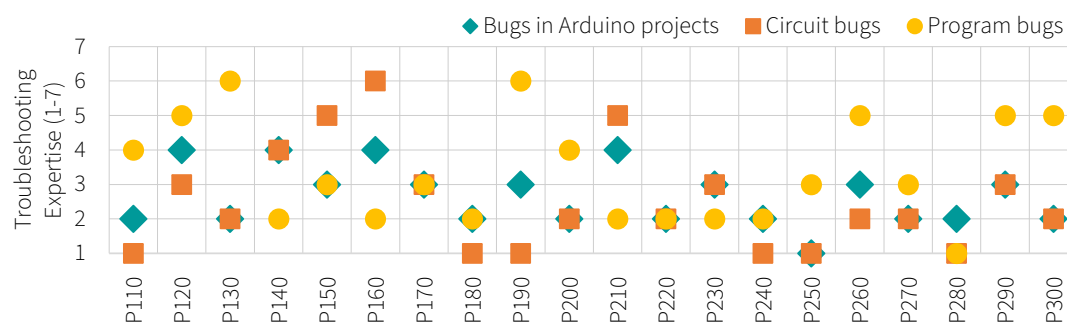


Figure 58. Individual participants' self-rated troubleshooting expertise: bugs in Arduino projects, circuit bugs and program bugs, from 1 (Complete beginner) to 7 (Complete expert)

6.2.4 Materials

6.2.4.1 Informed consent form

As with the previous studies, an Informed Consent form (Appendix Q) was created, to capture each participant's agreement to taking part, and their consent to data collection—including video recordings of the session—and the subsequent use of any data collected.

6.2.4.2 Background questionnaire

A background questionnaire (Appendix R) was created, to capture participants' personal and demographic information, as well as data about their experience and perceived expertise (self-rated) in physical computing (Arduino), programming and electronics, and whether they had received any external training or instruction, and if so, of what kind. This questionnaire was an adaption of the questionnaire used in Study 1A—questions were added about participants' perceptions of their expertise in troubleshooting 1) bugs in Arduino projects, 2) circuit bugs (in general, not just in Arduino), and 3) program bugs (likewise, in general).

6.2.4.3 Support Materials questionnaire

A questionnaire was created to capture participants' opinions of the Support Materials. (Appendix S). In this, participants were asked to rate the usefulness of different elements of the Support Materials, provide brief written feedback about any specific likes and dislikes, and indicate the extent of their agreement/disagreement with a number of statements in respect to perceived effectiveness, usability and fitness for purpose (see section 6.2.6.5 for further information about the data collected by this questionnaire, and how it was analysed).

6.2.4.4 Interview topic guide

I created a topic guide (Appendix U) to guide a semi-structured interview conducted at the end of the session. Questions aimed to elicit more detail about participants' perceptions of the effect/impact of the Support materials upon their troubleshooting, and to find out what they thought of them, thus supplementing and triangulating data captured via the Support Materials Questionnaire.

6.2.4.5 Troubleshooting tasks

A variety of materials were created for use in the tasks, including the Support Materials, two buggy prototypes, two task instruction sheets and a demo video of the desired prototype behaviour. Participants were also given access to a laptop and some additional equipment.

Support Materials

Chapter 5 describes the design and development of the card deck and supplementary materials (together referred to as the support tool, or Support Materials) used in this study, with section 5.5 (The Tactical Troubleshooting toolkit) describing each element in detail.

To summarise, the support materials used in this study comprised 36 Tactics cards, 6 Category cards, 6 Best Practice cards, the playmat and the cards stand.



Figure 59. Cards in card stand, and playmat

I decided not to include the component cards in this study. During a pilot run of the study, the end-user developer participant focused mainly on using those cards, even when repeatedly reminded to use the Tactics cards. As my main interest, for this study, was in respect to the Tactics cards, and the process of troubleshooting, the component cards were not included in the support materials.

For this study, the Support Materials also included an index card, containing a short set of rules. Rather than prescribing a specific method in detail, it stated that participants were required to use the cards, including the questions on the back, advised them to use the playmat, and reminded them of the 'diagnose → fix → evaluate' cycle. It also suggested that if participants got

stuck, they should take a random (i.e., any) card from the stand. No other rules were specified in respect to the use of the support materials.

Buggy prototypes

Two buggy prototypes were created: *Buggy Prototype A (BPA)* used in Task 1 and *Buggy Prototype B (BPB)* used in Task 2. Rather than counterbalancing the order of exposure to the prototypes as well as the support materials, I chose, instead, to ensure that the prototypes were equivalent in complexity/difficulty.

I chose to base the buggy prototypes on the same project used in Study 1—a simplified⁴ version of the ‘Love-O-Meter’ project in the official Arduino Starter Kit (‘Arduino Starter Kit’ n.d.)—because the study had provided much insight into the problems that end-user developers can encounter when building that particular project, including specific bugs that can be introduced during the process. Using the same base prototype per task meant that a participant would not potentially encounter circuits or components of differing familiarity to them, which might affect their performance and skew their opinion of the two tasks. Although it also meant that there was potential for a learning effect to affect performance, it was agreed, in discussion with my PhD supervisors, that the benefits outweighed this, and that my analysis could take it into account.

As the bugs observed in Study 1A had been introduced organically by end-user developers, I used a subset of these for the tasks in this study. A shortlist of potential bugs was created, in preparation for choosing a final six for use in the study. Using six different bugs, of varying levels of difficulty, in the study, meant that all participants would hopefully be presented with at least *some* challenge, irrespective of their previous experience or level of expertise.

How many bugs per buggy prototype?

Once it was agreed that tasks would involve the simplified Love-O-Meter, a small pilot study, involving two end-user developer participants, was conducted, to decide whether to use one circuit bug per prototype, or multiple circuit bugs per prototype in the study. Participant 1

⁴ The original project included code that converted the raw ADC (analog to digital conversion) readings (read from the analog pin), first to voltage and then Celsius. The program I used in this study included no conversion.

undertook six consecutive tasks, each involving a Love-O-Meter prototype preseeded with a single bug. Participant 2 undertook two tasks, each involving a prototype preseeded with three bugs. Each participant performed half of their tasks with the Support Materials and half without.

A very strong learning effect was observed for Participant 1, who, after managing to find and fix the first bug (i.e., successfully complete the first task) and then encountering the same prototype in the second task, merely replicated the working circuit (by comparing their memory of what ‘perfect’ looked like) for the remainder of the tasks—having only one bug to find and fix per prototype made this a relatively straightforward activity. I decided to minimise the learning effect by having multiple bugs per prototype.

Focus group with Arduino experts, to choose bug sets

To decide on the two sets of bugs with which to preseed the two prototypes, I conducted a two-hour focus group with six Arduino/physical computing experts from the Bristol University Interaction group—five had at some point been employed or commissioned to develop physical computing prototypes, three also had experience of *teaching* physical computing and all developed physical computing prototypes as part of their current work.

In this session I demonstrated a working Love-O-Meter, then presented each bug to the group, describing the fault(s) it consisted of, and the symptoms of it at runtime. The group then discussed and ranked the bugs in order of complexity / challenge, taking into account that the troubleshooting study would involve novice Arduino users. In preparation for the focus group, I had created a number of buggy prototypes, instantiating the bugs, which the experts were able to use in these discussions. A kit of spare components (LEDs, sensors, resistors) was also available, so that the experts could modify the prototypes, if they wished.

The experts collectively brainstormed possible equivalent combinations of three bugs per prototype, taking into account not only the complexity of individual bugs and the knowledge required to resolve them, but also the effect of the bugs in combination and how partial resolution of one or more bugs would affect runtime behaviour, i.e., symptoms of failure. Of these combinations (bug sets), the group selected two which they felt to be most equivalent.

These two bug sets were then hallway-tested (an impromptu, observed think-aloud debugging session involving a passing member of the department) and subsequently deemed equivalent in debugging complexity. The group also suggested the specific order in which these two buggy

prototypes should be exposed to the participants, to minimise potential carryover effect to the greatest degree possible.

The final buggy prototypes

Two buggy prototypes were created for use in the tasks. Both of these were based on the simplified ‘Love-O-Meter’ project in the official Arduino starter kit, which lights up three LEDs in a specific order, in response to readings of body temperature, when a temperature sensor is held between the fingers. The ‘ideal’ or model circuit and program for this project are described in detail in section 3.2.3.4.

Each buggy prototype was preseeded with three bugs—one of the two bug sets chosen by the focus group. As mentioned, all of these bugs had been introduced by one or more participants in the first study.

Buggy prototype A (Task 1)

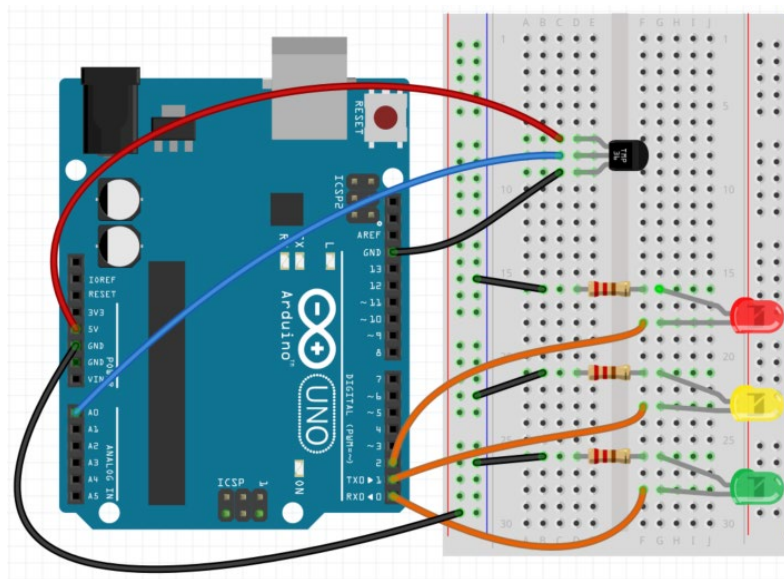


Figure 60. Buggy Prototype A (Task 1)

The bugs preseeded into this prototype were:

- A1. LEDs connected to RX/TX: Two of the LEDs (green and yellow) are connected to digital pins 0 and 1, which also transmit and receive data when serial communication is used at runtime.

- A2. LEDs the wrong way round: All three LEDs are seated the wrong way around in the breadboard. Their anodes are therefore connected to Ground and their cathodes are connected to digital pins.
- A3. Ground rail not connected to Ground pin. The breadboard rail (-, blue) set up to provide a shared Ground connection for the 3 LEDs is not connected to an Arduino Ground pin. Instead, the wire that comes from the Ground pin is connected to the adjacent rail (+, red).

All three of these bugs were visible to the eye if the participant knew what to look for, although for A2 the participant would have to look very closely at the LEDs. A2 is an extremely common bug—inserting an LED the wrong way around in a breadboard (i.e., incorrect orientation) is a very easy mistake, one which even experts make—a participant in study 1A described checking LED orientation as “the obvious thing to do”. A3 was also classed as an easy bug and could be found by simply tracing (following) the wire from Ground to the breadboard, or from the LEDs to the rail, i.e., checking for circuit completeness. A1 was the most complex bug. It required the participant knowing (or discovering) that the first two digital pins (0 and 1) have additional functions when serial communication is employed—additional labelling of those pins on the Arduino indicates that there is something different about them from the others.

Buggy prototype B (Task 2)

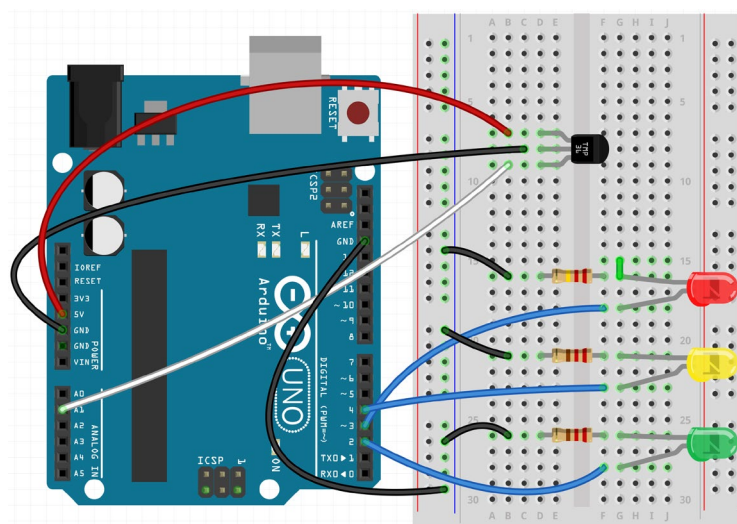


Figure 61. Buggy Prototype B (Task 2)

This prototype contained the following bugs:

- B1. LED resistor is too high a value: The resistor connected to the third (red) LED is 220K ohms instead of 220 ohms.
- B2. Sensor Ground and signal swapped: The sensor is miswired, with the Ground pin of the sensor connected to an Arduino analog pin (A1) instead of a Ground pin, while the signal (voltage out) pin of the sensor is connected to an Arduino Ground pin instead of an analog pin.
- B3. LED digital pins in non-consecutive order: Instead of the LEDs being connected to pins 2, 3, 4, in sequential order, the order of connection is pins 2, 4, 3.

Once again, all bugs in this prototype were visible to the eye. Many participants in Study 1 had miswired the sensor and struggled greatly with the diagnosis and resolution thereof—particularly when the prototype was completely built, as in these buggy prototypes—therefore B2 was considered to be the most difficult bug in this prototype. B3 could be discovered by tracing each of the wires from the LEDs to the digital pins, however I deliberately used the same colour of wire for each LED and wire lengths were very long, making this bug less visible. The resistor bug (B1) was a relatively easy bug to solve but with only one differently coloured band differentiating visually between the 220 and 220K resistors, the error might not be obvious to the untrained eye, particularly as resistors are quite small.

Note: to successfully complete a task, participants did not have to replicate the model circuit and/or program exactly but merely to modify the prototype so that the specification of correct runtime behaviour was met. This required resolving/fixing all preseeded bugs, but it would be up to the participant to decide what circuit or program changes to make in order to achieve that—some bugs could be fixed in different ways, for example, by changing the program or the circuit, or by changing different parts of the circuit.

Additional task resources

Two task instruction sheets were created (Appendix T)—a ‘With Support’ instruction sheet and a ‘Without Support’ instruction sheet. Each explained the task goal and specified the prototype runtime behaviour (how it should work) that would be used to judge task success. They also listed constraints within which the participant must operate, for example the amount of time allocated, and any do’s and don’ts, for example, a reminder to think-aloud. The ‘With Support’ task instruction sheet contained additional instructions/rules for the use of the support materials within that task.

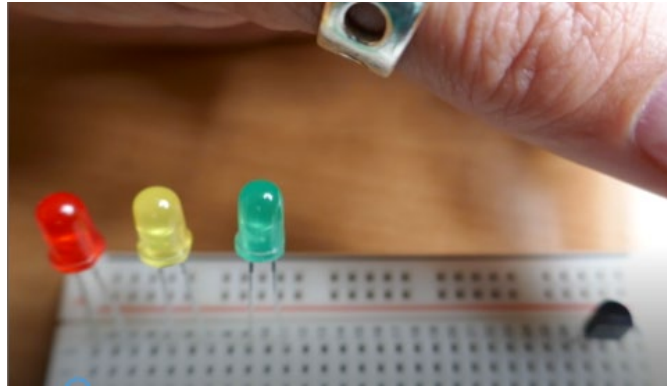


Figure 62. Still image from the video in which the correct prototype behaviour at runtime was demonstrated, showing the temperature sensor and LEDs, but none of the wiring.

A video was created to demonstrate the target runtime behaviour, i.e., how the prototype should behave when all bugs had been fixed—Figure 62 shows a still image from this video. It also showed participants how to interact with the sensor in order to trigger changes in LED state. The video frame was angled to show a hand interacting with the temperature sensor, and the resulting LED behaviour, but not any aspect of the construction or configuration of the circuit or program, for example the wiring. The same demonstration video was used for both tasks.

The microcontroller board used for the buggy prototypes was an official Arduino UNO R3 board. The IDE was the official Arduino IDE, and it was installed on a laptop computer running the Windows 10 operating system.

The parts kit was very similar to that provided in Study 1. It contained TMP36 temperature sensors, 3 colours of LEDs (red, yellow, green) and resistors in a wide range of values (4.7Ω , 220Ω , 330Ω , 560Ω , $1k\Omega$, $10k\Omega$, $1M\Omega$, $10M\Omega$). All component compartments were labelled and several of each component were provided. A spare USB cable and a digital multimeter was also provided, along with a selection of jumper wires of different lengths and colours.

6.2.5 Procedure

6.2.5.1 Overview of procedure

Each participant individually attended a two-hour-long session at a pre-arranged time convenient for them. The session sequence of activities is also shown in Figure 63.

After completing the background questionnaire, the participant undertook two hands-on troubleshooting tasks—Task 1 then Task 2—while thinking aloud (verbal protocol). In each task, the participant had a set amount of time to find and fix all bugs in a specific ‘buggy prototype’.

Each participant undertook one task with the support tool and one without. Participants were randomly assigned to one of two groups, determining for which task number (and therefore buggy prototype) they had the support materials. Table 20 shows the groups and order of conditions.

Table 20. Participant task groups and order of conditions

	Task 1	Task 2
Group	Buggy Prototype A	Buggy Prototype B
NSWS	No Support (NS)	With Support (WS)
WSNS	With Support (WS)	No Support (NS)

Before the ‘With Support’ task, the participant was given time to familiarise themselves with the support materials, and ask questions about them.

After the tasks, the participant completed the support materials questionnaire and, finally, was interviewed.

6.2.5.2 Sequence of activities

On entering the room, the participant was seated at a desk with a laptop on it. They were given a verbal overview of the session, in which they were also deliberately misinformed that the devices in the two task would be different. After signing the informed consent form, they filled in the background questionnaire.

Thereafter—the main part of the session—the participant undertook the two tasks, one after the other, the group to which they had been assigned determining whether they had the support materials for the first or second task.

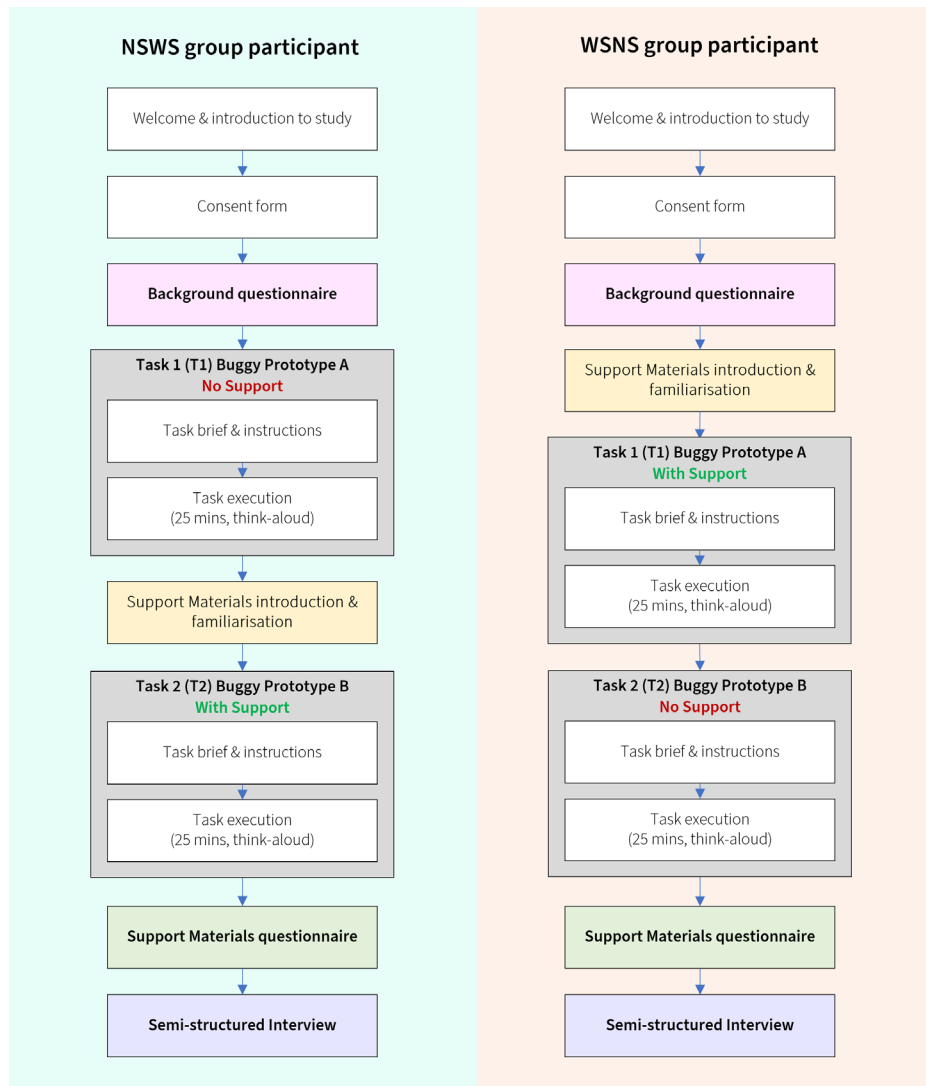


Figure 63. Session sequence of activities for participants in each of the two groups

Immediately prior to the ‘With Support’ task, the participant was shown the support materials. Following a verbal introduction to the information structure of both toolkit and cards, the participant was given dedicated time (up to fifteen minutes) to familiarise themselves with the support materials, while thinking aloud. During this time, the participant could also ask questions about the support materials.

Before the task timer was started, the participant was given the buggy prototype (although not yet allowed to inspect it) and the task instructions. The physical prototype (circuit) was connected to the laptop and placed on the desk in front of them, and the program was opened within the IDE on the laptop. They were taken through the task instructions verbally, and shown the demo video, ensuring that they understood the target runtime behaviour and what was

required of them. They were given no indication of where the bugs might be located—that is, in the circuit or program—or how many bugs were in each buggy prototype. The participant had access to the task instruction sheet and demo video throughout the task and were asked to keep the physical prototype (circuit) within an area marked on the desk.

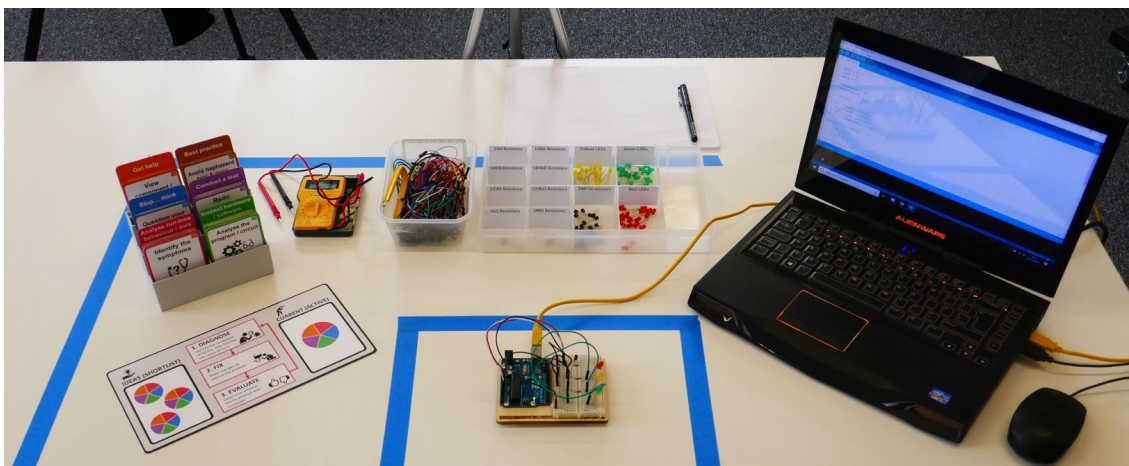


Figure 64. Setup for With Support task. The troubleshooting cards can be seen, in their stand, top left, and the playmat is bottom left. The buggy prototype is in a taped area, directly in front of the participant, with the parts kit above it.

For the ‘With Support’ task, the support materials were placed on the desk, to the left of the physical prototype (circuit), where they could be easily seen and were in close reach. The cards were in the stand, grouped within their categories, with the playmat directly in front of the stand. The participant was told that they *must* use the cards for the ‘With Support’ task, as I wanted to see what happened *when* participants used them, not *if* they used them, and that they *must* use the reflective questions on the rear of the cards, to guide their thinking when troubleshooting. During the ‘With Support’ task, if it became obvious that the participant was not using the support materials as required, I would remind them to do so, by holding up one of two notices—‘Use the cards’ or ‘Use the questions’, depending on what the participant was neglecting to do. The participant was encouraged to use the playmat, which was explained to them, but could use the cards however they wished. I advised them that if they had difficulty choosing a card to use, they should take a random card—i.e., randomly draw *any* card—from the stand.

In each task, the participant was given 25 minutes to find and fix bugs in the buggy prototype and asked to ‘think aloud’ while doing so—as well as general thoughts, they were specifically to articulate any troubleshooting-related thoughts. In addition to the laptop, they had a parts kit of spare components, cables and a multimeter. They had access to the internet, via the Chrome

browser on the laptop, and could also use the IDE's built-in help. They were not allowed to search for a project which matched the exact functional design of the buggy prototype—that is, a project which used readings from a temperature sensor to control the behaviour of LEDs—but had no other search constraints. They could ask me questions to clarify the brief, but could not ask for hints, advice, or anything else that might help them with the task.

The task stopped either at the end of the 25 minutes, or if the participant decided that they had met the brief—it was up to the participant to decide when they thought they had found and fixed all the bugs, based on the written specification and demo video. They were asked to demonstrate the final runtime behaviour and were not told how successful they had been in finding or fixing bugs.

After all tasks were completed, the participant completed the Support Materials questionnaire. This happened when both tasks were finished, rather than straight after the 'With Support' task, so that the participant would have experienced troubleshooting with and without the support materials and could factor any observed differences into their answers.

Finally, once the questionnaire had been completed, the debriefing interview was conducted.

6.2.6 Data collection and analysis

6.2.6.1 Video recordings

The main source of data about how participants thought and behaved during the tasks, as well as their feedback in the interviews, was in the form of video recordings. The sessions were video-recorded, using a combination of screen-recording software (to capture all on-screen activity), laptop webcam and three external video cameras—a close-up, birds-eye view of the prototype, a wider view of the desk and participant, and an over-the-shoulder view of the participant's use of the support materials. As in Study 1A, all videos recorded for each participant were pre-processed and synchronised to a single, composite, split-screen video, using Adobe Premiere video-editing software (Figure 65).

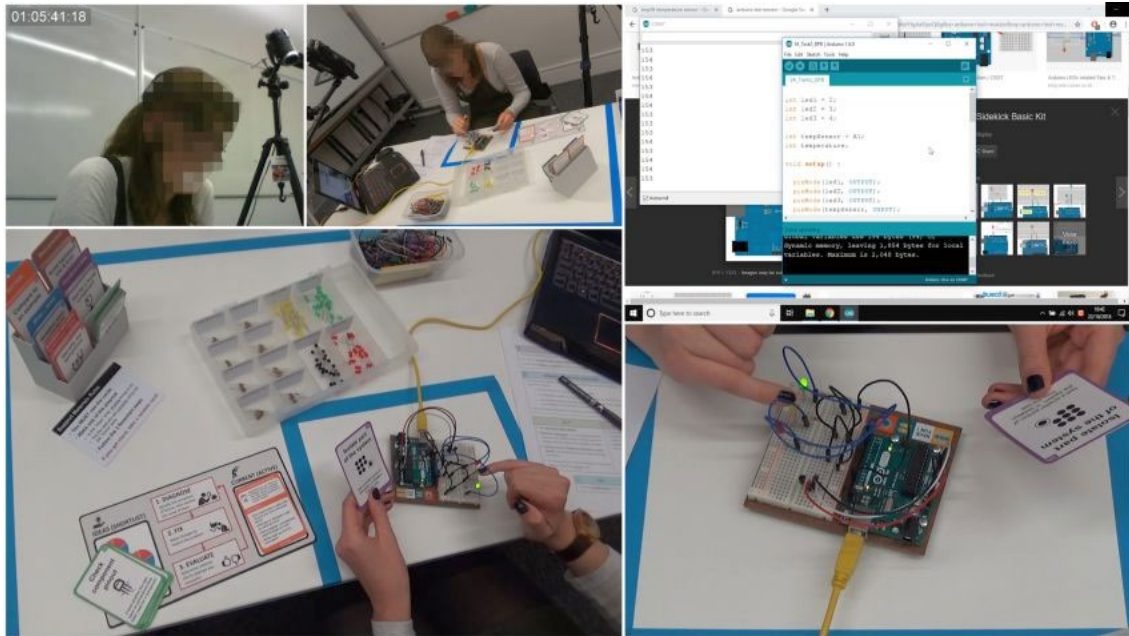


Figure 65. Still from a composite video of a participant task recording, showing in clockwise order from top left 1) the participant's head and shoulders view, 2) desk-facing view, 3) screen capture, 4) overhead view of the circuit and 5) over-the-shoulder view of support materials use

6.2.6.2 Participant-created task artefacts

After each session, digital photographs were taken of each buggy prototype (circuit), from several angles. As the buggy prototypes were reused for subsequent sessions, a digital representation of the final state (at task end) of each buggy prototype was created, using the Fritzing software application, recording its final, physical configuration, later double-checked for accuracy against the photographs. During this process, the buggy prototypes were scrutinised for circuit bugs—as well as recording these graphically within the resulting Fritzing layout images, they were also noted in writing. These bug data were later captured to a spreadsheet.

Programs edited—or created—by each participant were saved. Any bugs within them were noted and later captured to the same spreadsheet as the circuit bugs.

Any notes or diagrams created by the participant were digitally scanned.

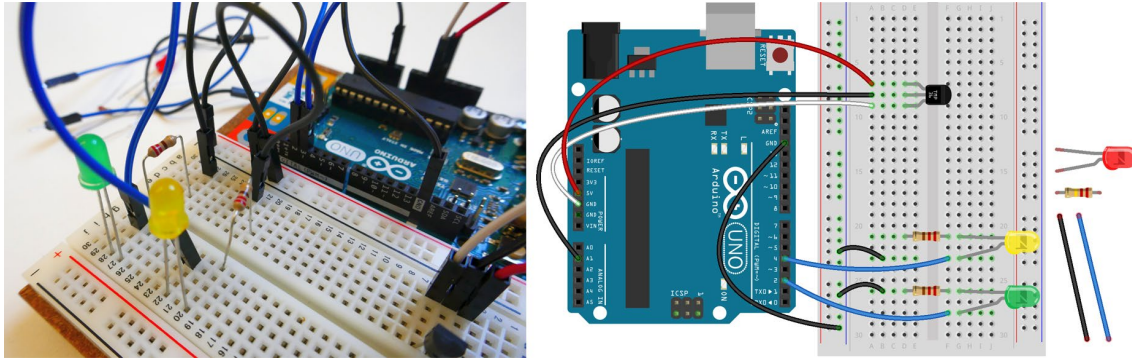


Figure 66. Photograph (left) and Fritzing image (right) capturing the final state of a prototype at the end of a task

6.2.6.3 Analysis: Task success

Based on the artefact and video data, participants' success in completing each task was recorded as a binary 1 (Yes) or 0 (No), within a spreadsheet. A participant was said to have completed the task *successfully* (task success = 1/yes) if, at the end of the task, the prototype behaved as specified in the task brief (instruction sheet and demonstration video) and the program and circuit contained no bugs that would interfere with correct (i.e., specified) behaviour. Task success was not based purely on whether the preseeded bugs had been resolved. If a participant had found and resolved all of the preseeded bugs in a buggy prototype but had introduced new bugs which still remained at the end of the task, they failed the task. The task programs and circuits were scrutinised after the sessions, for any bugs not obvious at runtime, and the details recorded to a spreadsheet. This, along with the facilitator notes made during the session and watching the task videos—specifically, the close-up view of runtime behaviour—was used to reliably determine task success.

I counted and compared the number of participants who achieved task success within each task (T1/T2), as well as how many achieved task success within each condition, i.e., with or without the support materials (NS/WS).

6.2.6.4 Analysis: Bugs fixed; Bugs remaining

When recording details of remaining bugs to a spreadsheet, the bug location (circuit or program) was noted, along with a very brief summary of the bug, i.e., what was incorrect. For each task, I noted whether each of the preseeded bugs had been resolved/fixed (Yes=1; No=0) and whether any *new* bugs remained in the circuit (Yes=1; No=0) or program (Yes=1; No=0). I did not count the number of new bugs remaining, only whether or not bugs remained.

I counted how many of the preseeded bugs had been fixed by each participant, per task (T1, T2), as well as how many had been fixed in each support condition (NS, WS). I also calculated how many participants had bugs (preseeded circuit bugs, new circuit bugs and new program bugs) remaining at the end of each task (T1/T2), also for each support condition (NS, WS).

6.2.6.5 Support materials questionnaire data

Data captured via the Support Materials Questionnaire (Appendix S) included the participants' ratings, on a seven-point scale—from Not at all useful (1) to Extremely useful (7)—of the usefulness of each of the different elements of the support materials (tactics, categories, playmat & rules, card stand), and of the card format as a medium for delivering support. Two freeform text questions captured, in brief, written feedback of anything the participant specifically did or did not like about the support materials. Finally, nineteen questions captured the extent of a participant's agreement or disagreement with a number of statements regarding the perceived effectiveness (eight questions), usability (seven questions) and fitness for purpose (4 questions) of the support materials, again on a seven-point scale—from Strongly disagree (1) to Strongly agree (7). Only the endpoints of the scales were labelled descriptively, and participants provided a rating by circling a number on the scale.

After the session, the paper questionnaires were digitally scanned and then the data entered into a spreadsheet, where it was summarised using descriptive statistics.

6.2.6.6 Debriefing interview data

The semi-structured debriefing interview, guided by the interview topic guide (Appendix U), was video recorded. Besides questions about participants' perceptions of the effect/impact of the support materials upon their troubleshooting, and what they thought of them, the interview also probed any interesting observations noted during the tasks.

A video sequence was extracted, from the composite video recordings, that covered the specific period from the point at which the participant began to fill in the Support Materials Questionnaire (as many participants had chosen to comment verbally on some of their ratings, explaining them) until the end of the debriefing interview, which immediately followed the questionnaire completion. I used Otter.ai (Otter.ai Inc. n.d.) to first automatically transcribe the audio tracks of these videos and then manually edited them for accuracy, creating a full record

of everything said during this part of the sessions. I downloaded these transcripts and imported them into MAXQDA (VERBI Software GmbH n.d.), along with the video files, linking and aligning each video with the appropriate verbal transcript. To familiarise myself these data, with a view to extracting themes from them through coding, I watched each video within MAXQDA while simultaneously reading the verbal transcript of it, occasionally amending the transcript with contextual information in brackets, where I felt it necessary, for example, if a participant pointed to something specific while speaking. As the intention of my coding was not theoretically driven, my coding process was loose and pragmatic, rather than following any strict or specific inductive or deductive method. I was interested in what people thought of the support tool, including specific components and design aspects of it, but also evidence of any thinking or behaviours it engendered, to enable me to assess whether—and if so, how—it had or had not met the aims behind its development, and identify any particular aspects of the tool that might be improved in a future iteration. As the designer of the support tool, I was mindful of potential bias when coding, taking care to ensure that the coding decisions I made would enable me to paint as accurate a picture as possible of participants' opinions of the tool and of its success in fulfilling its aims as a potential medium for scaffolding end-user developers' troubleshooting. Initial coding was guided by, but not limited to, the questions (and categories thereof) in the support materials questionnaire, the card design principles derived from the literature, and the aims and features of the support tool, but also revisiting and recoding previously coded transcripts when additional patterns began to emerge. Over several rounds of the dataset, I reviewed and refined my codes, merging and/or renaming them, and grouping them into categories appropriate to the developing themes.

6.3 Results

The results will be reported in respect to the research questions they answer, first looking at the effect, on task performance, of participants having used the support materials (RQ1) and then at participants' subjective opinions of the support materials (RQ2).

6.3.1 What effect do the cards have on helping end-user developers troubleshoot?

To answer this question, I looked at the effect of the support materials on the outcomes of participants' troubleshooting, that is:

- Whether participants managed to successfully complete the tasks (task success)
- Whether participants managed to find and resolve the preseeded bugs in each task
- Whether additional bugs had been introduced, into the circuit or program, which remained unresolved at the end of the tasks.

I compared the performance of the two groups (NSWS and WSNS), as well as, irrespective of group, performance with and without the support materials, in the two tasks.

6.3.1.1 Task success

A task was counted as *successfully completed* if, at the end of the task, the prototype contained no bugs that would prevent the prototype from behaving as specified, irrespective of whether the participant was aware of them.

Many participants struggled to successfully complete the tasks, particularly the first task. Figure 67 shows the number of participants in each support condition who achieved task success in each task. Only one participant completed Task 1 successfully, and did so without the support materials (NSWS group). One other participant, in the same group (NSWS), did manage to find and resolve all of the preseeded bugs in Task 1, while using the support materials, however, program bugs that they had introduced while troubleshooting (incorrect operators and temperature thresholds in their conditional statements) still remained at the end of the task.

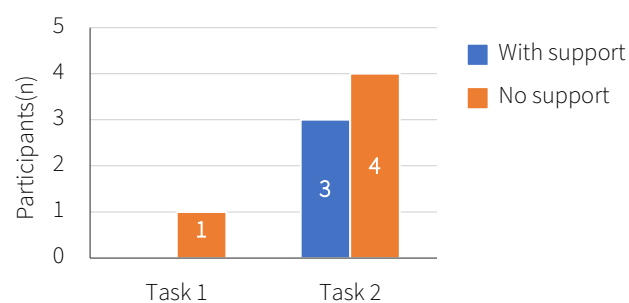


Figure 67. Participants who achieved task success, with and without the support materials, in each task

I had anticipated improvement for the second task, given participants' familiarity with the Love-O-Meter by this stage, and it was subsequently observed. Seven participants, in total, completed Task 2 successfully, slightly more without the support materials (4, NSWs) than with (3, WSNS).

Overall, each group successfully completed the same number of tasks (4), in total, and while slightly more tasks were successfully completed without the support materials (5), than with (3), the differences were small.

What this shows is that there was no obvious effect of the support materials upon task success. Many participants struggled, particularly in the first task, and while improvement was noted for the second task, as expected, there was little difference, overall, between the two groups or support conditions.

6.3.1.2 Bugs

I then looked at the bugs that participants resolved, firstly at the preseeded bugs.

Each buggy prototype contained three bugs for participants to find and fix. Overall, far more preseeded bugs were fixed in Task 2 (Figure 68). Again, I had expected some improvement, given the potential for a learning effect between the two tasks, i.e., the participants becoming familiar with the simplified Love-O-Meter prototype as a result of being exposed to it in Task 1.

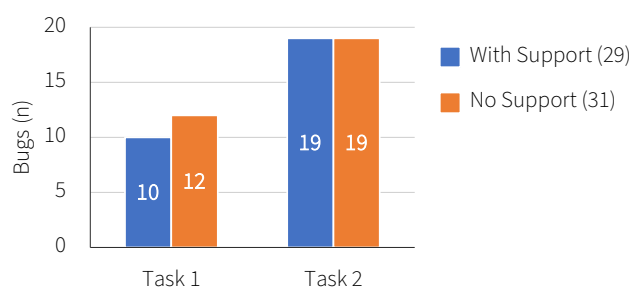


Figure 68. Preseeded bugs fixed with and without the support materials in each task

Overall, the group who had the support materials in the second task (NSWS), fixed slightly more preseeded bugs (31) than the group who had them in the first task (WSNS, 29) and, slightly more preseeded bugs were fixed when participants did not have access to the support materials (31) than when they did (29), however, again, the differences were small.

Specific bugs

I looked at the specific bugs that had been preseeded in each task, to see if particular groups had been more successful in fixing any of them.

There were no notable differences between groups (i.e. support conditions), for either task (Figure 69). While the A1 bug (LEDs connected to RX and TX pins)—classified as difficult in the focus group with Arduino experts—was fixed by only two participants, both in the NSW group and therefore troubleshooting without support materials, each of these participants mentioned having experienced this bug previously. Other than that, very similar numbers of participants in each group fixed each of the other bugs.

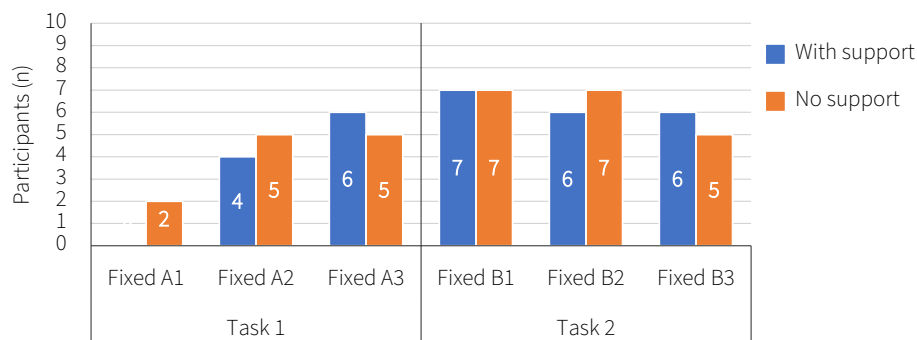


Figure 69. Participants (n), per support condition, who fixed specific preseeded bugs in each task. The WSNS group had the support materials in T1; NSW had them in T2.

Participants with bugs remaining—preseeded and new

I then looked at *all* bugs remaining at the end of each task, not just preseeded ones. As preseeded bugs were not described at the level of individual faults (i.e., connections to be rectified), I chose to count the *number of participants* with bugs remaining, rather than the number of bugs remaining—this meant that the same, unambiguous unit of measurement could be used for both preseeded and new bugs.

Some difference between groups is noted—however, there are no discernible patterns to this, and again, the difference is not large (Figure 70).

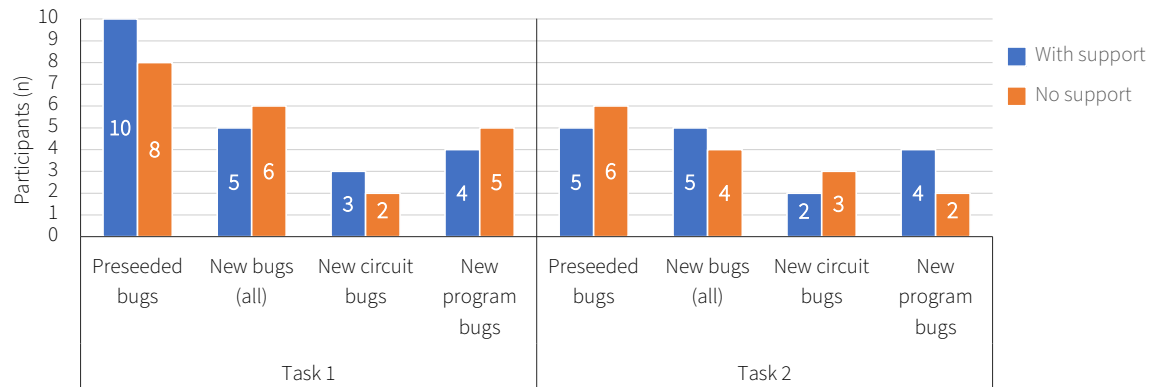


Figure 70. Participants (n) per support condition, with bugs (preseeded, new, new circuit bugs, new program bugs) remaining at the end of each task. (Note: 'New bugs' counts participants with any new bugs remaining, irrespective of their location)

Preseeded bugs were the most common type of bug remaining, particularly in Task 1 (18 participants).

In both tasks, participants in both groups had introduced—and subsequently failed to resolve—new circuit bugs and new program bugs. Almost half (9) of the participants had new program bugs remaining at the end of Task 1. This suggests that participants' troubleshooting led to them changing—incorrectly—parts of the program, as was also observed in the first study.

Looking at Figure 71, we note that more participants had new bugs and new program bugs remaining at the end of the tasks in which they *did not* have the support materials. While the difference in numbers is relatively small, this is encouraging. Conversely, slightly more participants had new circuit bugs remaining at the end of tasks in which they did have access to the support materials. Although disappointing, this may suggest that participants were at least making changes in the correct location, i.e., the circuit rather than the program, however, again numbers are small.

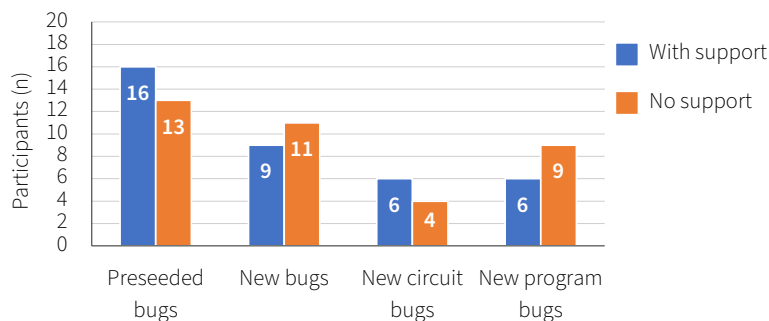


Figure 71. Participants (n) with bugs (pre-seeded, new bugs (all), new program bugs and new circuit bugs) remaining at the end of tasks with or without support

Level of success in fixing preseeded bugs

However, there is some evidence that participants were more likely to make at least *some* progress when they had the support materials, than when they did not.

I looked at what level of success participants had fixed the preseeded bugs, counting the bugs fixed, and putting these numbers into bands: All (3 fixed), Some (1 or 2 fixed), and None (0 fixed).

I noticed that overall (Figure 72), while fewer participants (4) had fixed all preseeded bugs when they had access to the support materials than when they did not (7), fewer participant (3) fixed *none* of the preseeded bugs (compared to 7 without support), i.e. more of them (17) had fixed at least 1 of the preseeded bugs when they had the support materials than when they did not (13).

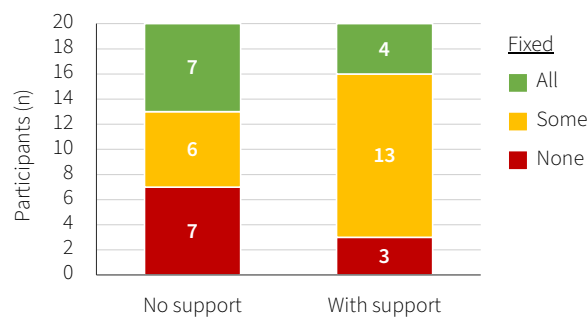


Figure 72. Participants (n) who fixed none, some or all of the preseeded bugs, with and without support.

Breaking this down further by task (Figure 73), I then noticed that only three participants in the WSNS group fixed none of the preseeded bugs in Task 1, when they had access to the support materials, compared to half of the participants in the NSWS group, who did not have them, while in Task 2, all participants who had access to the support materials fixed at least one preseeded bug. This suggests that the support tool may succeed in helping end-user developers to make at least *some* progress, even if they do not resolve all of their problems—more so than without.

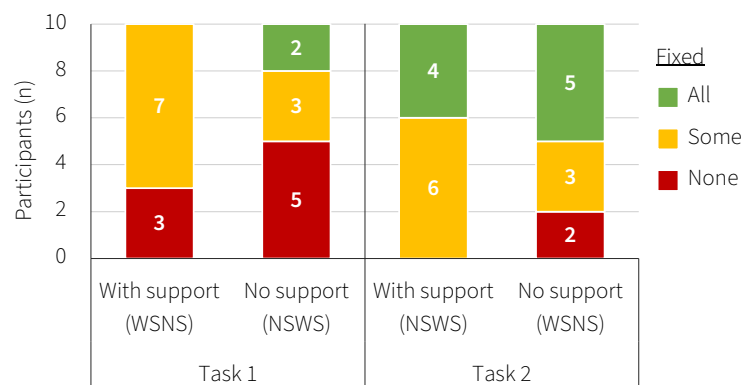


Figure 73. Participants (n) per support condition who fixed none, some or all of the preseeded bugs in each task

Also encouraging, is that most (8/10) of the participants, who had access to the support materials in Task 1 (WSNS), managed to fix at least one preseeded bug in Task 2, while half (5/10) of them managed to fix *all* of the preseeded bugs, even without access to the support materials in this task. This suggests perhaps there may be a learning effect associated with the support materials. Certainly, as I will shortly discuss, comments made by some participants in the debriefing interview support the suggestion that the support materials lead to learning that transfers to future troubleshooting, even when the support materials are not available (see *A positive priming / learning effect* in section 6.3.2.2 What did the support materials achieve?).

While none of WSNS group completed T1, the majority (7) made at least *some* progress. In T2 we see a dramatic improvement for this group, with half completing the task and three others fixing at least one bug. Only two participants failed to make any progress in this task.

Having looked at the outcomes of participants' use of the support materials (RQ1), I now turn my attention to participants' feedback on the design and use of them (RQ2).

6.3.2 How do end-user developers view the support tool?

To answer RQ2, I analysed the data collected via the Support Materials Questionnaire (SMQ) and the debriefing interview. In this section I report some of the key results from the questionnaire analysis, supplemented with thick descriptions of themes identified in the interviews, substantiated with participant quotes.

Ratings in the SMQ were captured using a 7-point scale and only the ends were labelled descriptively. When reporting results from the SMQ, I refer to anything above 4 (the midpoint) as agreement (with a statement or as being useful) and anything below 4 as disagreement. The midpoint is reported as neutral. Strong disagreement means a score of 1, strong agreement a score of 7.

This section is structured as follows:

- In section 6.3.2.1 I present *feedback about the design of the cards*, and other aspects of the support materials, structured as follows:
 - *The physical card format*
 - *Card content & design*
 - *Organising and using the cards*

- In section 6.3.2.2 I then discuss *what the support tool appeared to achieve*, based on feedback from participants, in terms of the aims of its development:
 - *Providing/prompting ideas*
 - *Making end-user developers think more when troubleshooting*
 - *A positive priming / learning effect*
 - *Recognition of value in changes to process*
 - *Suitability for novices*

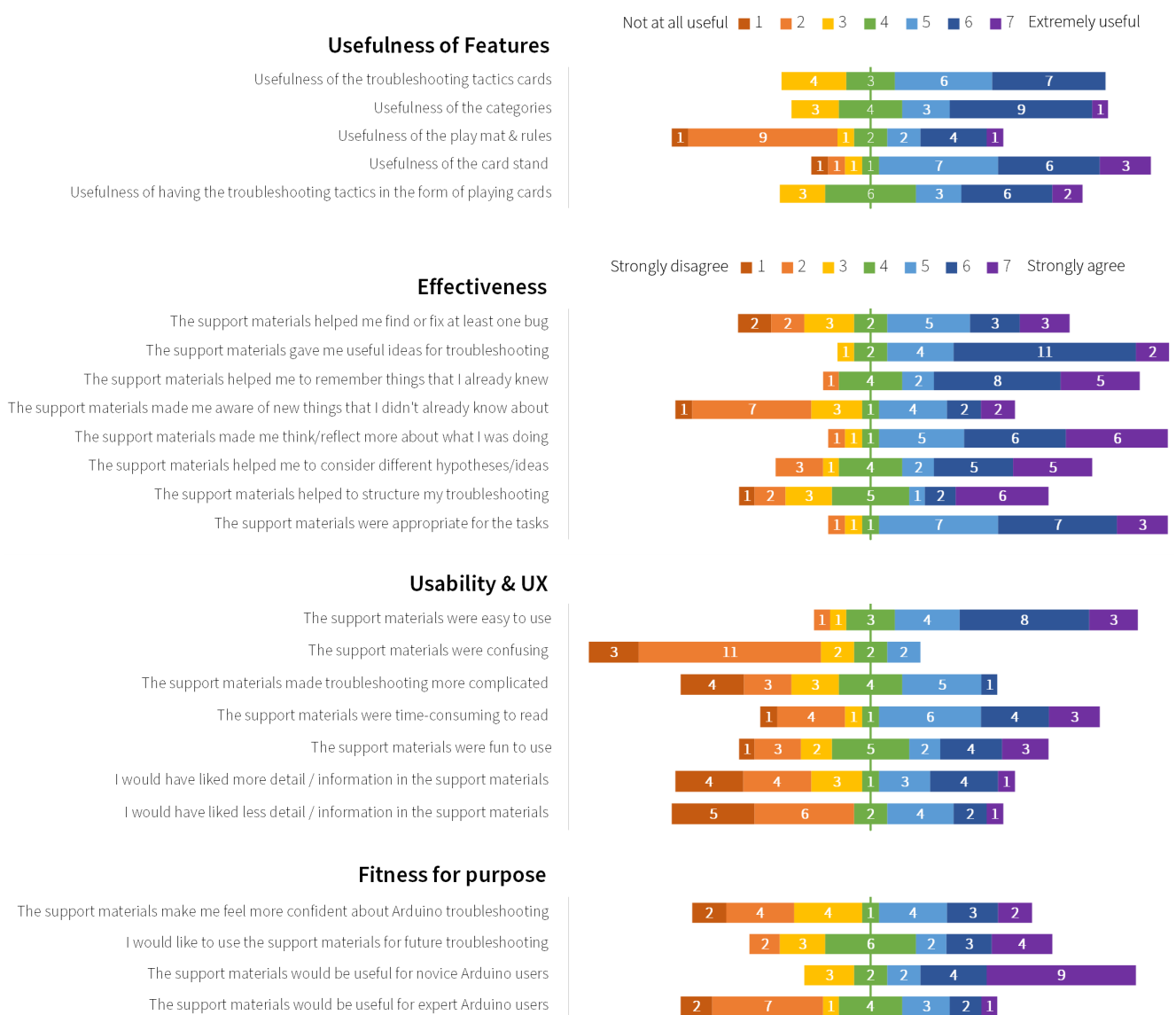


Figure 74. Support Materials questionnaire responses. Participants gave each question a rating from 1 to 7. The number within each coloured box represents the number of participants who chose that rating. Green represents the midpoint (rating = 4)

6.3.2.1 Feedback about the design of the cards

While the support materials, in the context of the study, comprised several different elements, the cards were the main vehicle through which support was provided, with the thirty-six tactics cards representing the primary form of content. In the SMQ, thirteen participants rated these as useful (Figure 74). The interviews provide insights into what participants did and did not like about the cards, and why.

The physical card format

Slightly more than half of the participants (11) found it useful to have troubleshooting tactics in the form of physical cards (Figure 74), with a further six participants feeling neutral about this. Several expressed their appreciation for the format, acknowledging some of the advantages discussed in section 0, and the benefits over having support in a different medium.

For example, P120 reaffirmed the flexibility of cards in a physical space. As has been noted in other studies, cards can be quickly and easily manipulated, as a kind of external cognition, which can also be helpful in tracking. This participant reflected that while comments in programs can be used to track what has been done, there is no analogue for *hardware*, and suggested that cards could provide this to some degree, potentially through additional areas on the playmat. They also felt the cards format to be less distracting than switching between hardware and software, enabling them to stay focused on their task:

“So that's why I'm proposing I can use the card as a record as well. So it's gonna be easier for me to manage. [...] I mean, in, in software, it's easy to do that because you comment things out and you look back [...] ‘Okay. I've done this for the hardware’.”
(P120)

“With this card, it's just putting it there (mimes putting a card into a pile). It's more helpful for me to concentrate, rather than switching between different media.” (P120)

Another perceived advantage of providing support in the form of physical cards was that it is neither software nor hardware based—P180 felt that this encouraged them to take a step back from their circuit and program. They suggested that screen-based support may even lead them to focus more on the program, which was something I had speculated about as a potential cause for the large number of program-related problems in the first study, and the frequent misdiagnosis of circuit bugs as program bugs.

“Having something to pick up and hold [...] it got me moving away.. 'What am I doing? Move away from that board, move away from that screen. [...] As opposed to 'Oh, it's on the screen. Let me just Google this'. Google is a distraction. If something else was screen-based, I could have thought 'Well as I'm here, let me just jump to this code. [...]’” (P180)

Some, however, were concerned about space that physical cards would require, for example, P160 would prefer to have them in digital form:

“I would have preferred if it was software based. [...] At home [...], I don't think I'll be able to have, like, space to put cards on and do all that (P160)

Card content & design

Some participants felt the card format to also have advantages for information delivery—presenting support content as smaller chunks of information/advice makes it easier to absorb, or more memorable:

“I do like that it's broken up into small pieces. I think that makes it more manageable and easier to take, like, individual suggestions.” (P230)

“Wise advice in a digestible format” (P140)

The ‘information iceberg’

Feedback validated the decision to *layer* information across two card sides, keeping the front very simple and consigning the bulk of the text to the rear:

“I like that you've used both sides so that there's less in one go. (P270)

“in a way this (points at the card title) is the tip of the iceberg. Actually, this side of the card, right? [the front] And this back is like what is underneath.” (P170)

Although participants were instructed to use the list of questions on the back of the cards to guide their thinking, they usually focused mainly on the front sides when reviewing and selecting cards. This confirms that distinct, comprehensible titles are crucial to effective use of the cards within an end-user developer troubleshooting workflow.

“Questions are helpful, but I think for me, even just having the bold reminder on the front is what I really needed more. It was just, like, the reminder to do.. like, the general idea.” (P230)

Participants’ views were mixed when it came to the amount of information on the cards—some found it acceptable or ideal, some wanted more, and others less (Figure 74), As people have

different information processing styles and reading abilities, this was unsurprising. While this study *required* participants to use the questions on the rear of the cards, in a more naturalistic setting, end-user developers would have agency to choose which—or how much—information on the cards to engage with, so double-sided cards have the potential to satisfy multiple styles.

Imagery

The use of icons on the cards was well-received. Several participants—including some with reading difficulties—felt the icons helped them to understand the concepts and spoke favourably about the simplicity of these images. As in my first study, some participants mentioned struggling to understand more technical imagery in other resources:

“I like that they were quite to the point. From the front of it, you could get what it was about. [...] instead of it just being text, having an actual image to tie it to, is good.”
(P210)

“What I did like was how the icons are more relatable to what I might understand than typical Arduino. [...] I don’t understand the.. especially in a circuit, they always mess up the circuit with the drawings and it doesn’t translate to me” (P280)

Visual puns—used for attention and to aid memorability—were also well-received, for example, P260 felt this made the cards feel more fun and approachable:

“They’re very funny. It’s interesting. Like writing ‘logging’ as a log. [...] I like that. It just makes you laugh [...] It just doesn’t take things seriously. I think it’s a big problem in any sort of engineering computer science field that people are just really boring.”
(P260)

The use of open questions

The chief aim behind delivering the main content as questions, rather than instructions or more direct signposting, was to encourage end-user developers to *think* more when troubleshooting, while simultaneously helping them to develop their problem-solving skills by highlighting lines of thought that might prove helpful. However, as I will discuss in section 6.3.2.2, not all were happy with this extra cognitive load. Participants’ views were fairly polarized regarding the use of questions. Many recognised why this approach had been taken, and found it useful, for example P180, who reflects that this is actually how they learn best, while even some of the participants who found the questions time consuming within the constraints of the study (e.g., P250 and P190) saw value in it.

“For me, this sort of thing works better than if someone had said, ‘Here are the instructions to build this circuit’. I will build the circuit, but I will not learn. I will simply follow directions. And that’s not the same.” (P180)

“Questions make you think about things a lot more [...] and then that’ll make you understand what you’re doing a lot more in general. [...] they prompted me to try and discover things that I didn’t know, which is kind of good. I guess that’s because they didn’t give away the answer but encouraged me to try and find it myself.” (P250)

“I think questions are good because you can’t apply an instruction to a lot of situations, but you can apply a question to lots of situations.” (P190)

However, others, such as P160, felt that concrete information would be more helpful than open questions. I had anticipated that some participants would prefer *direction* over *reflection*, from the pilot study and card design focus groups. Participants’ use of external help in my studies demonstrates end-user developers’ need for information about the tools and equipment they use. While this study focused on troubleshooting tactics, the Component cards (section 5.5.4) represent one way to satisfy some fundamental information needs of this type.

“It feels somewhat patronizing (laughs). I personally just prefer having information, like, saying for this temperature sensor ‘This is the ground, in, out. Make sure that the orientation is correct,’ rather than ‘Why does orientation matter?’” (P160)

Some difficulties with the questions were reported, for example, not being able to answer them made one participant feel “anxious”. As I later discuss, feelings of anxiety or stress regarding the support materials often seemed to be related to the pressure under which participants felt themselves to be in the study, but P280’s feedback suggests a need for additional routing or suggestions for end-user developers who struggle to make any headway using open questions:

“It’s not answering anything as well, it’s just providing more questions for me to.. that I don’t have answers for” (P280)

Tactic specificity

As I describe in section 5.5.1, tactics deliberately vary in specificity—the deck contains general problem-solving tactics as well as very specific ones, and while some tactics are very practical, others are deliberately more thought-provoking. Participants’ reactions to this also varied:

“I do like that some of these are very practical, sort of.. technical, sort of, approaches, whereas some of them are just about, you know, this ‘stop and think’ and just sort of using lateral thinking and, and sort of taking a step back from it, that type of thing” (P110)

"I think the level there is a bit of confusion in terms of, um... the things, whether it's specific or not. [...] this one is quite sort of general. And this one is sort of very specific"
(P120)

At least some confusion stemmed from focusing on certain aspects of the cards and not paying attention to others. P120 changed their opinion of one card upon taking a closer look at it. At first glance, this may not seem like a particularly important observation, however, it suggests that this end-user developer may have dismissed, as not relevant, tactics that may have been of use to them. It may be that time pressure played a part in this, however, it is still worth noting.

"I now read this, it's clear, I mean. I was reading mainly this [the title]. So, I wasn't paying attention on the specific words here [the description]." (P120)

One participant suggested splitting cards into 'general' and 'specific' categories. It is possible that differentiating, somehow, between levels of specificity might not only improve end-user developers' understanding of a particular support tool, but also of troubleshooting in general—a future study could explore this.

"Have one category that is just really specific and a whole category that is just more general, maybe. [...] maybe I want to look for really specific answers, for Arduino, I don't want to be bothered with more general problem-solving techniques." (P260)

Organising and using the cards

Encouragingly, most (15) agreed that the support materials were easy to use and most (16) disagreed that they were confusing (Figure 74),

Participants had no constraints regarding card organisation—they could work however they wished. While most (16) agreed that the stand was useful (Figure 74), a few felt it would be more helpful to have more tactics visible at once, for example:

"Where you can see everything as you think, it's more helpful for me [...] if something's more visual, I could maybe like look across and see like, 'Oh, yeah, I forgot to do that' or something." (P280)

More than half (13) of the participants agreed that the categories were useful (Figure 74) and the interviews provide some insight into differences in opinion. For example, P180 found the categories distinct and easy to understand—their interpretation matching the logic behind the grouping, whereas feedback from P160 suggests that not all end-user developers may find these categorisations understandable or useful:

“Analysing is different from inspecting. Analysing requires sort of a systematic judgment of behaviour, whereas inspecting is more what I was doing: ‘Is that circuit correct?’ So those two are definite. Conducting a test is different from analysing it. It’s the predecessor to analysing it. If you don’t conduct it, you can’t analyse.” (P180)

“A lot of the categories, to me they look the same. [...] if they were, like, a little bit more distinct, it would make more sense.” (P160)

Participants also had no constraints regarding how many cards they could use at any one time, and I had assumed that in some cases it would make sense to work with more than one. Correspondingly, P120 found it helpful to have cards from multiple categories active, for example, a card from the ‘*Conduct a test*’ category and a card from ‘*Analyse behaviour/data*’. Again, this illustrates yet another advantage of using physical cards, i.e., on-the-fly creation of value or meaning through combination—a technique common to the methods suggested by several creativity support card tools (section 0).

“when I bonded to one section or category, I find it a bit difficult. But then I realized, ‘Oh, there are things that I can combine’, like, there are kind of mixing things, I started to put them together and then that’s easier for me” (P120)

The playmat

Participants were encouraged to use the playmat, and advised to follow the Diagnose → Fix → Evaluate (DFE) cycle.

Participants’ opinions of the playmat were very polarised. Only seven rated the playmat and rules as useful; over half (11) disagreed (Figure 74).

The ease and convenience of using the physical cards for tracking has already been mentioned, with P120 suggesting extending the playmat to include even more areas. Similarly, another participant likened the playmat to the Kanban-style boards commonly used by software teams to plan and track their activities, with separate columns—usually some kind of variation on ‘to-do’, ‘in progress’, and ‘done’—to show the status of different tasks.

The Shortlist area of the playmat encouraged participants to filter or triage the cards for usefulness/relevance. Separating the consideration of options (i.e., *planning*) from *acting/doing* was seen as helpful by P290, while P280 actually found shortlisting enjoyable:

“The playmat was great, actually. [...] And I think it’s really useful to separate the process. ‘Right, okay, so what might be useful in this?’ If you have a quick look and then say ‘Ok, I’m gonna try that and try that and try that and try that.’” (P290)

*"The only thing that I found, like, quite fun was having this [playmat] and having to pick out the ones I'm going to use. I was like structuring my to do list or something."
(P280)*

A few felt that the playmat was useful for keeping them focused and on track. Having cards put aside on the playmat prompted P280 to revisit the tactics they had planned to use, while for P180 it had benefits beyond the practical, keeping them calm in the face of time-pressure:

*"I did like the playmat. That was useful in the sense that it makes you put aside what you have to think now. [...] that was good. Because if it's sitting around there, it's like forcing me to look through it again."
(P280)*

*"The mat [...] was about 'Where am I'. It was really useful. [...] Again, keeping track of where I was, you know? It's 'What am I doing? I am doing this'. And also, with that, I was monitoring my thinking process, as opposed to racing ahead going Aaaaaaa' [...] And it stopped me panicking. Because I was aware of the time."
(P180)*

Others, however, were less positive about the playmat, for example, P230 and P190 both felt that the cards would be enough by themselves:

*"I don't entirely understand the purpose, I think, of the playmat. [...] I do like how it reminds you of just the general process, but as far as having a shortlist and active, I think that most people can kinda make makeshift piles on their own."
(P230)*

*"I felt like I was using the mat just because you told me to. But I think the cards on their own probably do a better job. I wasn't listening or looking at that [the DFE cycle] but I was subconsciously doing that anyway."
(P190)*

Although shortlisting was suggested, not enforced, some felt it an unnecessary hindrance—a time-consuming extra step that slowed progress. A few already had an idea of what they wanted to do—as soon as possible; for others, shortlisting conflicted with their usual way of working.

*"I prefer having all of them there and just reading all of them. [...] I prefer a mess (mimes dealing cards in front of them; laughs). For example, I might go through them and just put them on the desk and then read through them one by one rather than having an idea shortlist"
(P160)*

Two participants, including P200, did not like having additional items like the playmat on their work surface:

*"I'm not a 'having things out' kind of person. I like things up (points up, in front). Or there (points front left)... there (points front right). But this kind of desk chaos makes me slightly uneasy."
(P200)*

Method: Where to start? Where to go next?

Despite most (15) rating the support materials as easy to use, one issue for a few participants was the lack of guidance or signposting in selecting which cards to use. Although cards were categorised, there was no indication of a starting point, or cards to prioritise, for example, in the early stages of troubleshooting. While some participants did not appear troubled by this, for others it presented a challenge, for example, neither P110 nor P200 were sure how to correlate the cards with their problems, so relied, in part, on random card draw, to help them choose:

“I didn't understand the basic fundamentals of how it's supposed to work. [...] it wasn't immediately obvious to me which category I sort of needed to look at. So I was picking them a bit at random, and just hoping that something would come out of there that would trigger a chain of events.” (P110)

“Pick a random card [...] there was almost a bit of relief when I read it. I was like 'Oh actually, at least, I know, I'll just take one and then I can work with that'.” (P200)

More than one participant, including P260, felt that a more structured method, or some kind of signposting between cards would have been useful, while two participants, including P210, suggested that familiarity and experience would mitigate this issue.

“I think the cards don't really connect to each other. I look at the card and I think it would be great if it tells me 'Oh, also look at this. If you've done this, do this. And if you've done this, do this.' It kind of gives you different pathways through troubleshooting. Other than that, it's just like 'Okay, I have these cards, but this didn't work, which card do I take next?’” (P260)

“Now I wouldn't have that problem, you know? And the other thing is, I would probably start to have my own methods [...] I'd probably start to have cards that I start off with, that might not be the same cards that other people start off with. So I think that only applies really to your first time using it. [...] It's just not being used to it” (P210)

This feedback echoes what some other researchers have observed—while some users are happy to work with cards in a more freeform manner, or devise their own methods, others would prefer, or benefit from, a more structured method (Lockton et al. 2009). However, as the support tool is aimed at novices, it seems worth exploring whether it would help to provide a starting point—at least, some suggestions for good (or common) tactics to begin with—and/or, some kind of linking between cards, to aid navigation within the deck.

6.3.2.2 What did the support materials achieve?

I will now describe some of the benefits observed within this particular sample in terms of the aim of the support tool. They can be summarised as follows, and will be discussed in this order:

- Providing/prompting ideas, including reminders
- Making end-user developers think more when troubleshooting
- A positive priming / learning effect
- Recognition of value in changes to process
- Suitability for novices

Providing/prompting ideas

In the first study, I observed some participants running out of ideas, repeating unsuccessful diagnosis tactics, resorting to speculative changes or even giving up troubleshooting. A primary aim of the support tool was, therefore, to provide novice end-user developers with plenty of options to consider—hence the large number of tactics in the deck. Encouragingly, feedback from participants suggests that this was achieved, as most (17) agreed that the support materials gave them ‘*useful ideas for troubleshooting*’ (Figure 74).

As earlier discussed (section 2.8), troubleshooting, particularly, problem *diagnosis*, is a hypothesis-driven activity, and poor or incorrect hypotheses can not only hinder progress, but also cause further problems. For example, while end-user developers often look for external help when troubleshooting, they do not always know what to search for—as observed in my first study, where poorly worded or incorrect searches led participants to wrong or misleading information. P190 felt the tactics cards offer an advantage when seeking help, by providing readymade suggestions:

“When you look something up in a book or online, you have to give the query. So, in that sense, cards are useful, because when you’re not sure what you’re looking for, you can go there to know what to look for.” (P190)

Feedback from some participants appears to support the hypothesis in section 6.3.1.2, that participants were *less likely to make no progress*, when they had the support materials. For example, P110 almost gave up part-way through Task 1 (No Support), despairing at being completely out of ideas, and eventually resorting to unsuccessful speculative fixes that resulted in new bugs. In Task 2, however, the tactics cards helped them overcome similar humps. Feeling

at a loss can dent self-efficacy, which in turn affects motivation and resilience in problem solving (Bandura 1978). If end-user developers are less likely to be completely stuck when using the cards, this may have positive implications for their perseverance in troubleshooting.

“when something doesn't work in Arduino, I think I have this feeling of helplessness, because I'm aware that I don't really know what's going on. [...] I was completely lost. I was just.. I was looking for things lining up. But beyond that, [...] I didn't have much of a clue. So these were useful because at least they gave me things to try and different aspects.. different things to look at” (P110)

Another technique for getting participants ‘unstuck’ was the suggestion to take a random (i.e., any) card, which several participants found helpful.

sometimes I just get to a point and I'm like, 'Okay, I'm done'. [...] But it gives you that nudge to say 'Actually, is it facing the right way around? Did you think of that? Did you actually think that maybe your temperature sensor has the wires crossed? No, you didn't.'” (P180)

Many creativity-support card decks employ random card draw as a device not just to elicit ideas but also *novel* or *different* ideas—an effect which was recognised in this study too. P210 (NSWS) felt the cards would have helped them in the first task, while others also noted that the questions—and the cards in general—prompted different thinking and behaviours, with positive outcomes:

“I think it would just got me out my thought pattern. And stopped me thinking down the path I normally think.” (P210)

“What I liked about them is that they offer questions which you might not have thought of previously, which kind of help you progress.” (P250)

“they were useful mostly in directing my attention towards an aspect or a side of things that I was ignoring previously, you know, they were good at just 'Oh, yeah, I should take a look at that'. And in a couple of those circumstances, I was thinking, 'Oh, I've seen something wrong here.' So, I don't know how long how long that would have taken me without that prompting, to have noticed that myself.” (P110)

Sometimes chains of events led to discovery, while P210 felt that even just having cards visible on the mat prompted further ideas:

“I sort of started looking at something else and then while I was looking at that, I sort of saw some.. and you know, so I think that sometimes they indirectly led to me discovering something” (P110)

“often this happened, I would have one card. And that card would lead me on to another card. And if I just put that card back with the card that led me on, I would

forget about it, but to have it down, and like, you know, there, is good. [...] it was just like, you know, a jumping board” (P210)

Reminders of existing knowledge

If their development sessions are sporadic/infrequent—more likely, outside of the routine and obligation of professional practice—less-experienced end-user developers are further disadvantaged in troubleshooting. They not only lack the enhanced knowledge that comes with more experience, but extended periods of time between development can degrade previously acquired knowledge, making it slower or more difficult to retrieve, and increasing the potential for misapplication. For example, it had been over five years since P300 last used Arduino:

“I was confused with the connections and things and was like ‘Oh this is like connected right or wrong?’ [...] I thought that things could come to me naturally, but I was like, ‘Oh, is it right?’ ‘Is it in the right place?’” (P300)

Providing end-user developers with *reminders* was another aim of the cards and most (15) participants agreed that this was successful (Figure 74). It sometimes made a big difference to how well-equipped participants were to diagnose or evaluate behaviour. For example, in Task 2, the cards reminded P290 of the existence of the IDE’s Serial Monitor, and by using this, they were able to view the sensor readings, which had been hidden from them in the first task. Similarly, P190 was reminded of a common testing tactic which they hadn’t yet considered:

“I totally forgot that was an option. So, when I saw that [card] I was like ‘Ah, of course!’” (P290)

“swapping in for a faulty one [...], that would probably occur to me at some point, but it didn’t occur to me right away. But the cards made me think about it on the spot.” (P190)

Timely reminders sometimes led directly to participants locating and fixing bugs:

“I knew that from day one, that a circuit has to be complete. But just the fact that [...] I read that, made me think, ‘Hang on, let me just check if they’re all going to the same ground point’. And then it was ‘Oh no, they’re not’.” (P140)

And sometimes, as I will discuss next, even just a reminder to take a step back was valuable:

“‘Stop, think’ is a good reminder that maybe you’ve been trying too hard to fix something in one way.” (P260)

Making end-user developers think more when troubleshooting

Encouraging end-user developers to be *more thoughtful* troubleshooters was another key aim of the tool, informed by the empirical work described in the previous chapter. Several mechanisms were used to support this, chiefly 1) open questions on the tactic cards, and 2) the ‘Stop... Think’ category of tactics—all promoting *thinking* rather than *doing*. Additionally, by encouraging participants to *shortlist* tactics, I had hoped to get them to reflect on different options and make conscious choices, before acting.

Most participants (17) agreed with the statement “the support materials made me *think/reflect more about what I was doing*” (Figure 74), with six giving this the highest rating. Slightly fewer (12) agreed that the support materials made them “*consider different hypotheses/ideas*”. As many tactics could be viewed as potential hypotheses, it is surprising that more did not agree. Nonetheless, very few (4) disagreed and five gave it the highest rating.

As discussed, participants’ opinions varied regarding the use of open questions. But while some found using them time-consuming (see section 6.4), most were unable to avoid them entirely (although some tried to), with evident effect on their thinking:

‘they were definitely kind of forcing me to think a lot more about what I was doing. I don’t think I would have been thinking so much about, like, smaller things if I wasn’t reading the cards, because they were asking so many questions, so it kind of led me to.. made me more determined to try and figure out what was wrong with it.’ (P250)

Several mentioned finding the ‘Stop... Think’ category helpful, as well as specific cards within it, particularly ‘Question your Assumptions’, which seemed to resonate with many:

“the ‘Stop.. think’ card was actually really helpful. Because I think this is something I would.. I like to do, to over-obsess about a certain thing. And then after, sometimes hours, realize that it was a really easy thing to fix. I just didn’t take a step back.” (P260)

However, not everyone found the extra thinking and reflection comfortable. For example, P170 felt more conscious of their troubleshooting process—usually more of a trial-and-error approach—and questioning this made them feel less confident about troubleshooting—perhaps a temporary effect of the study, but still worth noting. And rather than seeing the thinking prompted by the support materials as useful to the troubleshooting process, P140 felt it distracted them from *actual* troubleshooting:

“it made me consider my own thought process and how I approach troubleshooting Arduino, my knowledge. So in a way, it was more difficult because I was less confident

of the process itself [...] it sort of collapse with my own process, which is a little bit more chaotic, and I try and fail, you know .. it's more about trial and error. [...] And with these I was all the time questioning [...] So, it made it slower in the sense of having to consider what I should do and questioning my method” (P170)

“they did make me think about what I was doing. But in the moment, I don't think that was helpful. [...] It's like asking an athlete, in the moment of throwing the javelin, 'Where's your elbow now?' [...] they don't think about it, but the moment you ask them to, it's probably gonna just throw them off.” (P140)

A positive priming / learning effect

Several participants who had the support materials in the first task reported a positive carryover, or learning effect, to the second task. This may be one reason for the large improvement in task performance for this group (section 6.3.1). For example, P200 felt that the knowledge they gained from the cards in task 1 had *more* impact on task 2—reflecting upon the ‘*Undo failed fixes*’ card had made them conscious of their tendency to *not* do this, so they monitored their behaviour more closely than usual in task 2, potentially avoiding new program bugs:

“it really stuck with me as something that's a bit of a weakness in the way I approach things. And so when I was looking at changing the numbers there and doing all of that, I was much more careful, I think, to make sure that I put things back to how they were, than I would have been otherwise.” (P200)

Both they and P140 reported getting most benefit from the time spent familiarising themselves with the cards before they undertook their first task, rather than actually using them in the task:

“I think learning it and then applying, is genuinely more meaningful for me than when I've got the (mimes holding the cards in front of them) [...] I actually think they're most helpful when I've looked at them and then I'm working without them.” (P200)

“the fact that I was made to read through them before doing either of the tasks helped both of them. And I think that was the more useful part of it than having to do it during.” (P140)

A *priming* effect was reported by P240 as well, with the cards putting them in the right state of mind and reminding them of their existing troubleshooting knowledge after a lengthy period of not having using Arduino. This translated into tangible benefit in the second task. While the support tool has been designed for use *during* troubleshooting, feedback from these participants suggests an additional use, that may even be preferred by some end-user developers—frontloading the information in the cards, whether to remind them of tactics in advance, or just to get them into a troubleshooting state of mind.

“because I looked at them beforehand as well, it sort of reminded me of all the things I'm actually meant to do when troubleshooting. Because some of this I've obviously learned already.” [...] (P240)

“it's almost like, you know, you just want to look through them regularly, you know, and just sort of keep getting your brain going. Yeah, they're almost like warm-up exercises. [...] To get you in the right brain state, a problem-solving brain state.” (P240)

Recognition of value in changes to process

Using the support materials meant, for most participants, a change in troubleshooting process. Using the cards—enforced to the greatest degree possible; some participants required several reminders—required a degree of thinking prior to action, which was out of character for some. As I have discussed, the playmat supported this process, encouraging participants to shortlist potentially useful tactics and then to choose at least one tactic card to work with, while simultaneously reminding them to follow the Diagnose → Fix → Evaluate cycle.

Structure

Opinions were mixed on whether the support materials helped to structure troubleshooting. nine participants agreed—six strongly—while six disagreed and five were neutral (Figure 74). In the interviews, several participants spoke of the support materials *providing structure* and *encouraging structured thinking*—both positive boons. For some participants, including P210 and P150, there was a noticeable difference between the two tasks, with their thinking and behaviour becoming more structured and on track when using the support materials:

“really useful to give you a direction and just a framework, really, to help you make sense of what you're trying to do. I guess a lot of people work very intuitively. But that can sometimes be in a sort of spiral type of fashion. This seemed to help in my thought process a lot.” (P290)

“it helped me also to structure things, because I had to, like, go through them. And then ‘Which one am I actually using first?’. [...] these helped me to say ‘No, I'm doing this one first, and then I'm doing this. I'm not checking the code before I've checked whether all the connections are right’. [...] In that sense it's really good because it helps you organize your thoughts.’ (P240)

“They kept me organized in what I was doing. I didn't branch off to different things. [...] I put stuff down and returned to it. I think when I was doing the first one, it was a lot more disorganized, I would start troubleshooting one thing and then I'd notice one problem, and then forget that I was troubleshooting one thing and then not come back to it.” (P210)

“It definitely made me more structured, in what I was thinking. [...] In the first task, when I started, I was thinking quite chaotically. It was like, 'Do I jump to software now? Should I check hardware? Should I check this?' [...] But like with this, it was sort of step-by-step, and structured.” (P150)

Transformative reflection

When reflection upon an experience or new knowledge includes evidence of a changed or different perspective, Fleck and Fitzpatrick refer to this as *transformative reflection* (Fleck and Fitzpatrick 2010). Several participants’ demonstrated this in the interviews, indicating that they had *learned* from the experience and felt their usual behaviour needed to change. In some this involved some recognition of the value of thinking prior to action. For example, for P180, the recognition of weakness in their existing troubleshooting process was revelatory.

“[it was] preventing me going down rabbit holes. [...]. It pulled me back on track. I did one thing at a time when I had the cards. I didn't get ahead of myself, thinking 'Oh, let me change this, this, this and this.' I kept pulling it back to 'check your assumptions'. One thing at a time. [...] my instinctive thing is 'give me that fucker'. [...] I start going in, all balls blazing. Having the cards forced me to slow down. Think. Proceed with logic. [...] without the cards, I just ripped the whole thing out and started wiring it again. [...] I didn't do that when I had the cards. [...] So that was really, really useful. [...] I've done it, the rip it up, start again, for years. I needed an intervention stopping me from doing that.[...] it was really interesting, looking at my process compared to a more structured process. And a real good insight into how I just react instead of act. I don't analyse. I don't think, I just react.” (P180)

P170’s reflection upon their usual troubleshooting process led them to realise that they usually stick to what they know, avoiding more ‘technical’ practices. Previously assuming their lack of Arduino knowledge to be their main shortcoming, using the support tool led them to recognise that their troubleshooting knowledge itself could improve:

“I fly through things basically [...] like the more technical parts of the process I usually don't consider because I don't know how to, or I'm not that familiar because I'm not that expert. [...] it's the same as debugging, when you're coding something. I never debug because, it's like, I don't know how that works, right? So I just.. if it doesn't work, maybe I spend two days trying to figure out what I've done wrong [...] And that these [the cards] is, in a way, to say like, 'Okay, maybe it's not that you're not an expert [in Arduino].” (P170)

Other participants who struggled with the cards also took something away from the experience. For example, both P110 and P220 came to the realisation that their current approach to learning

Arduino is failing to develop their knowledge or understanding, with significant impact on their ability to troubleshoot problems.

"I learned a lot. [...] I learnt that I should not rely on readymade exercises. [...] if I put away the book, and try to do the exercise on my own I will learn much more, because I will do so many mistakes that I took for granted, doing, following the steps [...] And by doing so many mistakes, that's how I'll learn. Because following the steps might take me half an hour to finish, but doing it blindfolded, it might take me a day. But I will learn much more." (P220)

"just copying exactly what someone else is doing is maybe not such a great way of learning Arduino because you're not really learning. You're just seeing what works, but not really understanding why. [...] I think you've exposed a crucial vulnerability, you know, a crucial gap in my in my knowledge." (P110)

Speculative changes

"I'm not always sure what I'm doing. So I'm always trying, 'let me try this and let me try that'. [...] It's because I'm not sure. I'm not confident enough to know exactly what I'm doing." (P220)

When used thoughtfully and deliberately, rather than by default or as a last resort, speculative changes can provide value within a troubleshooting process. However, in the first study, as observed in studies of non-expert programmers, some end-user developers made speculative changes in lieu of more thoughtful diagnosis, many of these resulting in new bugs. I hoped that encouraging participants to think more when troubleshooting would lead to fewer *haphazard* speculative changes, also explicitly warned against in the Best Practice cards.

Several participants noticed a difference in their behaviour in this regard. For example, P130, a self-confessed 'tinkerer', observed a change to their approach in the second task (With Support), as did P250, who noticed that they made far fewer changes to their program. This feedback suggests that the support materials may help to tackle a sometimes-destructive troubleshooting behaviour that is characteristic of non-experts.

"Every time I got stuck at a point, I would just look over and see if there's anything that would help me. And then if I saw something, then I'd immediately jump back into it. [Facilitator: What would you do normally?] I'd tinker around to, like, think of another solution." (P130)

"I know that I didn't change as much in the code, which I did do in the first one. So, yeah, maybe it led me to think more about my actions, so I didn't feel like I needed to speculate. I'd thought about it more." (P250)

Suitability for novices

Encouragingly, most (15) participants agreed that the SM would be useful for novice Arduino users, nine strongly so; only three participants disagreed (Figure 74).

One of the many reasons for choosing the medium of cards to provide support was the potential for it to encapsulate useful troubleshooting knowledge in easily digestible pieces. Additionally, I wondered whether cards might be seen as a little more playful than traditional forms of support and therefore appeal to non-experts. Both points were reaffirmed by P230, who thought the format to be particularly good for young people, like them, who might be resistant to reading lengthy guides and be more receptive to something simpler and a bit more fun. Likewise, P260 found them “cute” and “very approachable”, with the “game-like” feel a welcome counterpoint to conventional forms of guidance.

“especially for young people who are just starting to learn [Arduino], it draws your attention, it's easy to get through and like I say, I do like that it's broken into very small pieces. I think that makes it much simpler. And they're just a little more fun.” (P230)

“It's very game-like. [...] It's very different from what I would expect. It breaks like this kind of very dry, normal set of programming where you just read black and white long texts [...] It's a more fun approach maybe to troubleshooting or programming” (P260)

Although scaffolding end-user developers' troubleshooting is a primary aim of the support tool, informally educating novice end-user developers through using it is also of interest, similar to the Idea Garden, which helps end-user programmers become better problem solvers (Cao 2013). P150 felt that the support tool teaches valuable best practice and ways of thinking that are usually imparted by teachers, or gained through experience, while P160 felt the practical knowledge in the cards could be immensely valuable to novices:

“going through the disciplines that this sort of teaches you is very good. [...] all of this is about thought process. [...] It's just about how you think. It's literally these three steps (points at the flowchart on the playmat). Yeah. And people fail to do a lot of this when they try to do stuff, when they first begin.” (P150)

“when I was a very, very beginner, I knew how to code on this (points at the IDE), but I had no clue what the Serial Monitor was. [...] So, obviously, for a beginner, for a novice, they might know how to output and how to connect an Arduino to the circuitry, but they might not know how useful a Serial Monitor is, or where it is even.” (P160)

Ideally, think P200 and P290, the knowledge encapsulated in the cards would eventually be fully assimilated, through repeated use, until no longer needed:

“If you were to get to the point where you just naturally apply all of those principles to what they're doing, that's the kind of, I guess, the best end result.” (P200)

“I think you'd get used to it and you'd get to know it. It's the kind of thing I'm wondering if you'd use very intensively to start with and then not need it anymore.” (P290)

However, as discussed, one challenge facing end-user developers, particularly novices, is that the physical computing knowledge they *do* build up can get lost, or degrade, over time if it is not used. The learning/priming effect of the cards, even when perused upfront, rather than only in the act of troubleshooting, has the potential to address this. As well as providing reminders *during* troubleshooting, both P240 and P170 felt the cards would be a good way to refresh their knowledge when returning to Arduino after some time, as happened when they participated in this study. For P170 this has concrete, practical benefits, while in P240's case, the reminder that they *do* already have knowledge also served to boost their confidence and self-efficacy:

“I hadn't touched an Arduino in a year. [...] I don't even remember how the breadboard works. That's why my knowledge is spread throughout time. And then when I get back it's like, 'Oh my god, I need to go again to the internet because I don't remember how this works. [...] And this is basically a reminder of that. Like, you know, remember that this works this way. Remember that you have to think of these things. It's like, 'Yeah, right, so I'm in the Arduino world, again'” (P170)

“They definitely make me feel more confident. [...] for me, this is a really important one [...]. It's more about getting myself in the right state for something like that, right? I need to feel I know, already, a lot. 'I can do this', right? [...] I like this, because it makes me feel confident that I've got a brain and that I can solve problems” (P240)

As a novice end-user developer, P120 even feels that the awareness—suggested by the large number of tactics cards—that there are numerous options/ideas yet to explore, is comforting, and acts as a kind of emotional support, while P240 also reports finding a degree of comfort and support in the presence of the cards, not just the knowledge they impart. As novice end-user developers can find troubleshooting stressful, the suggestion that the cards may also provide this type of support, in addition to practical scaffolding, is very encouraging.

“I feel they get me kind of comfortable in a way. ' [...] I've still got options here' (laughs). Like, 'I can do things here, don't worry'. It's sort of like, uh, support from an emotional way. [...] I still have choices.” (P120)

“But it's more like, for me to surround myself with stuff I know (lays out several of the Inspect cards). This is how I normally approach stuff. (Exhales loudly) Safe. Calm. [...]

Sometimes I just need to watch a Schiffman video just to relax, you know, get myself in the right state of doing coding or whatever. And the same way, these cards just get you in that state.. and I think that's a really important role of the cards here. They could also be like little objects, little guardians (laughs).” (P240)

6.4 Discussion

I will now expand upon some of the frustrations expressed by participants, before finishing the chapter with a summary of the study findings.

6.4.1 Task-related frustration

As already mentioned, several participants found *having to* use the support materials frustrating.

*“They're very wise advice but it was an annoyance to have to keep referring to them.”
(P140)*

Shortlisting was encouraged, rather than required, but even participants who chose not to shortlist still had to use the cards. They had been given some time upfront to familiarise themselves with the card deck, but with no signposting as to which cards might be relevant or more/most useful, it was up to the participants themselves to determine this. How easy or quick they found the process of assessing cards depended on factors such as their existing knowledge and how much of a card they needed to read in order to make a judgement, further impacted by their reading ability/speed. And, as established, using the support materials also gave participants more to think about, as I intended.

For P220, who had rated their expertise in all types of development as low, the extra cognitive load of choosing cards from an unfamiliar deck, then working out how to apply them felt overwhelming:

“there's a lot. I mean, there's six categories there. And that takes time and, you know, trying to categorize.. your problems. Is it the orange? Brown? It's already enough in my brain trying to figure out the circuitry[...] And then trying to figure out the colour scheme, and the categorization, the titles and then there's the one side of the card and then the other side of the card, and the other side of the card is split in two. So there's a lot of things going on that need more dedication from me, that I can't afford right now, because I'm trying to concentrate on the circuit. (P220)

Several participants spoke about the impact that the time constraint had on their card use. It is obvious that there was a difference between how participants wanted—or planned—to use the cards, and what actually happened.

“At the beginning I put the cards out so that they'd give me ideas. And then at the end I didn't have time to go through half of them” (P300)

“It was quite time consuming to read them. That's why I was kind of, like, going through and very quickly discarding ones I didn't think were useful.. but if it wasn't a time-pressured task, then maybe I would read them more.” (P270)

Several also reported feeling that the cards might be more useful/effective under more naturalistic circumstances, in which they felt less pressure and had greater agency.

“If there was like a smaller amount and I was stuck on something, and I needed a prompt, I think I genuinely would use them and they'd be very useful. I think just because of the conditions it was in, it was just a bit more tense.” (P250)

“I just feel like I had too little time to really use these cards. I think my experience is relatively negative because it was kind of a hindrance in this scenario. Because I would operate quicker, maybe, or differently, just with my own set of rules.” (P260)

“I felt that I was under time pressure. [...] having to read the support materials, then thinking about them, then applying them to my project was a little bit difficult. Because usually when I read, it takes me a while to read and then process, then apply. [...] But if it was, like, me at home, if I had these, or even if it was on the computer, I would be like, skimming through them, maybe, like, the title may come to me and I would be like, 'Oh, yes', and then go back to it.” (P160)

It seems that for some participants, at least, the circumstances and constraints of the study led to a more difficult, and therefore possibly more negative, experience of the support tool—a limitation of this study that should be acknowledged. This and further limitations of the study are discussed in section 7.2.

6.4.2 Summary

This was a study to evaluate the novel support tool, described in Chapter 5. It aimed to answer two research questions:

RQ1: What effect does a physical card-based support tool have on end-user developers' success in troubleshooting circuit bugs in physical computing prototypes?

RQ2: How do end-user developers view the physical card-based support tool, having used it to troubleshoot circuit bugs in physical computing prototypes?

To answer RQ1 I analysed participants performance in two hands-on troubleshooting tasks, comparing the outcome of participants' troubleshooting with and without the support tool. I found little difference in measures of task success, the number of preseeded bugs resolved, *which* preseeded bugs were resolved, or the number and location of bugs, including new ones, remaining at the end of the task. However, I noticed that more participants made at least *some* progress—fixed at least one bug—when they had access to the support tool, suggesting that it may reduce the likelihood of end-user developers being completely stuck—making no progress—when troubleshooting. More work would be required to establish this with certainty.

To answer RQ2, I analysed participants' subjective feedback about the tool, collected via the support materials questionnaire and a debriefing interview. Key points from this analysis include:

- Many found the physical card format useful. Some felt it to be potentially less distracting than software-based support, although a few would have preferred a digital tool.
- Participants confirmed some of the benefits of cards seen in other projects, for example, flexibility in ways of working, and the creating of new meaning through spatial arrangement. Grouping cards also provided a convenient way for participants to track their troubleshooting.
- Card content was seen as useful and relevant to the tasks. The informality of the cards was seen by some, as a fun and accessible way to approach an otherwise dry topic.
- The information design has the potential to accommodate different reading abilities and information-processing styles. Instantiating content as small chunks of information, makes it easy to digest, compared to other support media, e.g., lengthy texts.
- Many found open questions useful, as a way to prompt thinking and independent problem solving, although some wanted more instruction/direction, or the addition of concrete facts or signposting.
- The tactics succeeded in giving participants useful ideas for troubleshooting. They helped to remind them of their existing troubleshooting knowledge and were seen as a good way to refresh degraded knowledge—a common problem for end-user developers. This had practical benefits in the moment, but also served to increase some participants' feelings of confidence and self-efficacy in their troubleshooting abilities.

- Some participants even found the cards comforting, providing reassurance that they still had plenty of troubleshooting options available. This perception of the cards as emotional support was an unforeseen benefit.
- Participants also reported the cards having a positive priming or carryover effect—from upfront familiarization with the deck, or having used them in the previous task.
- The cards did succeed in encouraging participants to think or reflect more when troubleshooting, and the ‘Stop... think’ category, resonated with some participants as a reminder to step back from ‘doing’ and consider other lines of thought. However not all participants were comfortable with having to think more, and felt it distracted them from ‘actual’ troubleshooting.
- Some participants would prefer a starting point, or signposting in selecting cards. Random card draw was helpful in getting some participants ‘unstuck’, leading them to areas of investigation they had not considered.
- Participants’ opinions of the playmat were polarised. Some found the playmat and shortlisting useful, as a way to focus thinking, and plan/structure troubleshooting. It helped them stay on track and avoid some of their usual pitfalls—several observed thinking and behaving less ‘chaotically’ than they did without the support materials, making fewer speculative changes. However, others found the playmat a hindrance and the act of shortlisting time-consuming—having to think upfront conflicted with their usual approach, for example, trial-and-error tinkering.
- Some participants showed evidence of transformative reflection—using the support tool made them aware of shortcomings in their existing process.

In summary, although the cards format did not suit all participants, and certain aspects of the tool were polarising (for example, the playmat), participants were mostly positive about the support tool, particularly the cards, or specific aspects of them, and felt them to be useful for novice end-user developers using Arduino. Feedback also suggests that the support tool met a number of the key aims behind its development, including:

- Making participants think/reflect more during troubleshooting
- Providing participants with useful ideas for things to try—giving them options and getting them unstuck
- Reminding participants of their existing knowledge
- Encouraging a change in process that some participants found helpful

Additionally, it was suggested that the cards may have a positive learning or priming effect, which may benefit future tasks, although more work would be required in order to establish this.

While some participants found it frustrating to have to use the support materials, particularly within the constraints of the study, feedback paints a fairly promising picture of the potential for support in the form of physical cards to help some end-user developers troubleshoot physical computing bugs.

This chapter concludes the detailed description of the work that I have undertaken during the course of my PhD research. In the next and final chapter of this thesis, I will summarise this work, including the key findings, and situate it in respect to the literature.

Chapter 7

Discussion and conclusion

To recall, the overarching aim of the work I have described in this thesis was to answer the following research question:

How can end-user developers be supported in overcoming problems they experience when developing physical computing artefacts?

To answer this, I first needed to increase our knowledge of end-user developers in this domain—the problems that they experience and whether they are successful in overcoming them, how they go about solving those problems, and whether their behaviours are effective. This was accomplished through two pieces of empirical work, observing end-user developers developing an Arduino-based physical computing device to a given specification—data analysis first focusing upon problems observed, and then upon troubleshooting of the most significant problems. By addressing these knowledge gaps, I was able to determine the support most needed by end-user developers when troubleshooting in this domain. I instantiated this support through the design and development of a novel, card-based tool for scaffolding end-user developers' troubleshooting, which I then trialled in a study with novice end-user developers.

In the remainder of this chapter, which concludes this thesis, I will first revisit my findings from the research undertaken, and highlight the contributions to the literature, including relationships to previous work. Thereafter, I will discuss some limitations of my research, reflect upon the approach and methods I adopted in this work, and the decision to focus specifically on troubleshooting, and conclude with an outline of opportunities for future work.

7.1 Contributions

A number of contributions resulted from this work. I will now summarise these in respect to the four primary research questions posed in the Introduction chapter of this thesis.

7.1.1 Contribution 1

Empirically grounded knowledge of the problems encountered by adult end-user developers when constructing and programming physical computing artefacts, including the frequency and location of problem types, and which problems are most likely to cause task failure.

This contribution was achieved by addressing the following:

TRQ1: What problems do end-user developers experience when developing a physical computing artefact? (Chapter 3, Study 1A)

This study resulting in this contribution builds upon my work prior to this thesis (Booth and Stumpf 2013), now investigating problems which arise when end-user developers construct and program a physical computing device from scratch.

To address this research question, I conducted an empirical study involving twenty end-user developers—Arduino users of varying background and expertise—who undertook a hands-on task, developing an Arduino-based physical computing prototype to a given specification. The first analysis of the data collected (Chapter 3, Study 1A) revealed the following key findings:

- All participants in this study, irrespective of their previous experience, encountered problems during the development task: all experienced obstacles (barriers to overcome), while most experienced breakdowns (errors in thought or action) and introduced bugs (faults) in their circuit, program, or both.
- While most problems occurred in respect to *programming* the device, the majority of task failures were primarily due to *circuit*-related problems—10 of the 14 participants who failed the task did so due to errors in circuit construction.
- Circuit-related task failures were mainly attributed to two types of bugs: *Miswiring*, for example, providing the wrong connections from the Arduino board to the sensor, and

missing components, for example, failing to use resistors with LEDs. Participants had serious difficulties localising these faults.

- In diagnosing the symptoms of circuit bugs that they had introduced, participants did not always realize that the fault(s) lay in the circuit and incorrectly tried to resolve the problem by modifying the program, leading to new program bugs; In other cases, participants misjudged which part of the circuit contained the fault(s), and modified a *different* part of the circuit, again, leading to new circuit bugs. This pattern of misdiagnosis resulting in new bugs proved fatal to task success for several participants.
- Background factors such as self-efficacy and self-rated expertise did not predict whether participants would successfully complete the task, nor the number, types and locations of problems they experienced.

Decades of work has shown that novice and end-user programmers experience problems when creating and modifying programs. This study provides the first evidence that end-user developers experience problems when developing physical computing devices too, and that some of these problems have grave implications for development success. From this study, we now know where problems are most likely to occur, which problems are most prone to misdiagnosis, leading to the new problems and more likely to prevent end-user developers from successfully building a working physical computing device. As in previous research investigating the problems of novice and end-user programmers (e.g., du Boulay 1989; Spohrer and Soloway 1986; Lahtinen, Ala-Mutka, and Järvinen 2005), we see that end-user developers in this domain experience difficulties with some of the most fundamental aspects of development—for example, declaring and using variables, and, specific to this domain, the wiring of LEDs or sensors in simple circuits. Similarly, participants’ difficulties understanding runtime behaviour—i.e. obstacles involving both program and circuit— echoes previous work too, for example, Ko and colleagues’ (Ko, Myers, and Aung 2004) finding that Understanding barriers are particularly challenging for end-user programmers and them lead to chains of further problems that lead to bugs is similar to the patterns of misdiagnosis I report here. This study also seems to confirm the previous suggestion that end-user developers can face even more challenges when trying to resolve physical computing problems due it involving both programming *and* circuits (Tetteroo, Soute, and Markopoulos 2013). Finally, research by others since publishing this work in Booth et al. 2016, has confirmed similar findings to mine, for students new to Arduino (Sadler, Shluzas, and Blikstein 2017; DesPortes and DiSalvo 2019), however, to my knowledge, my work was the first to address this area. While the sample is small, a search of the official Arduino forums for

'Love-O-Meter' turns up numerous posts describing many of the same problems that participants in this study experienced. This suggests that the participants and their problems are not atypical and that the findings will generalize.

Understanding the problems faced by end-user developers is crucial step towards determining how to support them. My next contribution—knowing what troubleshooting behaviours they naturally employ, and whether these are effective—takes us one step closer.

7.1.2 Contribution 2

Empirically-grounded knowledge of the natural troubleshooting behaviours of end-user developers troubleshooting failure resulting from *circuit bugs*—the type of bug observed to have most significant impact on task success—and suggestions for support that might benefit end-user developers when troubleshooting these types of bugs.

This was achieved through a study addressing the following question:

**TRQ2: How do end-user developers troubleshoot the most significant problems that arise during development, and from what support might they benefit?
(Chapter 4, Study 1B)**

A second, deeper analysis of data collected in Study 1A, this study sought to address a lack of knowledge about how end-user developers troubleshoot circuit bugs, and whether their approaches are effective. Building upon previous work in end-user programming, I identify tactics used by end-user developers when attempting to diagnose and resolve circuit bug-related problems, and show that as in other domains, end-user developers often use unproductive or destructive tactics when troubleshooting physical computing problems. One advantage of this study was that I could observe end-user developers troubleshooting bugs that they themselves had introduced while developing a physical computing prototype. Even though the study was in a controlled environment and participants had to complete their physical prototype within a specified time, their behaviour was arguably more naturalistic than it would have been in a formal debugging study where participants are required to troubleshoot preseeded bugs. In particular, I was able to observe how participants troubleshot a variety of bugs, some of which I might never have anticipated. Key findings include:

- Participants used a number of tactics to diagnose their problems, fix their bugs and evaluate their fixes, however, these were not always effective.
- Lack of domain knowledge often led to poor or incorrect hypotheses, for example when participants tried to interpret runtime behaviour or output—a frequently adopted tactic—and reason backwards from as to the cause of failure.
- Participants frequently sought external help, but sometimes conducted poor/incorrect searches, or had difficulty understanding, judging or applying what they had found. Some also made mistakes when using resources—over half of the episodes in which participants copied examples introduced new bugs.
- Inspection was frequently observed, however, poor or incorrect hypotheses led some to inspect parts of their prototypes that did not contain faults and make incorrect assessments regarding its correctness.
- Participants struggling to diagnose, or running out of ideas, often resorted to making speculative changes. Far more speculative changes were made by those who failed to complete the circuit, and these types of changes resulted in over three times more bugs than they fixed. In some cases, participants failed to reverse these, compounding their problems. In contrast, when participants localised their bugs, the changes they made resulted in far more fixes and far fewer bugs.
- When participants were stuck, they sometimes stopped troubleshooting and continued developing while bugs still remained, adding further complexity to their prototypes, making subsequent diagnosis and fault localisation even more difficult.
- Direct feedback in the location of a fault did help participants to localise some bugs, for example, the sensor heating up due to miswiring.
- Prototype complexity played a role in whether participants were able to successfully diagnose their problems. Bugs in simple circuits were often more easily localised and resolved, however, when a prototype contained multiple dependencies, participants struggle to diagnose failure, as runtime behaviour/output was more difficult to interpret.
- Incrementally constructing and testing their prototypes, or reducing dependencies through tactics such as isolation, increased the likelihood of end-user developers being able to localise and fix their bugs successfully, however, not many who failed to resolve their circuit bugs adopted these approaches.

Overall, participants' difficulties in troubleshooting their problems are consistent with the literature, including considerable work attesting to debugging being particularly challenging for

non-experts (e.g., Lahtinen, Ala-Mutka, and Järvinen 2005). Similarly, some of the above findings have been observed in studies of novice and end-user programmers, for example, the tendencies for non-experts to make speculative changes and introduce new bugs (e.g., Perkins et al. 1986; Cao et al. 2010; Gugerty and Olson 1986; Nanja and Cook 1987).

While novice and end-user programmer behaviours when debugging have been studied by several other researchers (e.g., Katz and Anderson 1987; Cao et al. 2010), I believe this work to be the first to have looked at these in the context of *physical computing*, specifically for *adult, end-user developers*. Some of the tactics observed in this study have also been seen in studies of the strategies of end-user programmers, for example, *Inspection*, and seeking *Help*, however, some behaviours, for example, swapping in a component of exactly the same type and specification, or changing the spatial orientation of a component, have no equivalent in programming that I can think of. My work therefore complements the work on end-user programmers' debugging behaviours, extending it into a new domain—troubleshooting of problems within physical computing development, specifically hardware bugs.

Based on these findings, I propose that support for end-user developers' troubleshooting should focus on 1) help with general approaches that will help them to become better troubleshooters, and 2) specific support that will aid them in diagnosing the cause of bug-related problems, fixing bugs, and in evaluating the success of bug fixes (section 4.4.2).

Support I suggest for specific aspects of troubleshooting includes:

- **Planning and hypothesising:** Considering and prioritising different hypotheses and tactics, making more thoughtful decisions regarding what action to take.
- **Recognising and defining failure:** Better identification and analysis of the symptoms caused by bugs, in order to generate better hypotheses.
- **Focused analysis of runtime behaviour/output:** Guidance in the types of analyses that can be used to diagnose failure and evaluate the result of fixes.
- **Problem decomposition:** Ways to break a problem down or simplify it, e.g., reducing dependencies through tactics such as isolation, to establish the boundaries of failure.
- **Focused testing instead of haphazard speculative changes:** Approaching speculative changes in a more thoughtful way, as focused tests driven by hypotheses, ideally with a clear idea of what to look for in the results; suggestions of tests to perform.

- **Thorough inspection:** Awareness of the types of visual checks—including for common errors—that can or should be performed before making changes.
- **Incremental, iterative progress:** Incrementally building and testing prototypes; making one change at a time, and immediately evaluating the result.
- **Dealing with failed fixes—**Reversing changes that did not resolve failure, rather than building further upon them.
- **Following an iterative process:** Performing thorough diagnosis before a fix attempt, and then immediately evaluating whether the fix was successful.

General principles for supporting end-user developers include:

- **Encourage thinking/reflection:** Thinking through their problems will help end-user developers become better troubleshooters Support hypothesis generation and prompt reflection—before, during and after action.
- **Support perseverance with systematic troubleshooting:** Reduce the risk of end-user-developers giving up or making speculative changes, by providing them with troubleshooting ideas and a process to follow.
- **Encourage and support planning and tracking of troubleshooting:** Help end-user developers to consider and carry out necessary steps and remember what they have tried.

The next contribution draws on contributions 1 and 2, instantiating the suggestions for supporting end-user developers in the form of a novel support tool

7.1.3 Contribution 3

A novel, physical card-based tool to support novice end-user developers when they are troubleshooting physical computing problems, particularly circuit bugs.

This contribution resulted from addressing the following research question:

TRQ3: How can we design a deck of physical cards to support end-user developers in troubleshooting physical computing problems, particularly circuit bugs? (Chapter 5)

In Chapter 5 I described the design and development of my third contribution: a novel, physical card-based troubleshooting support tool for novice end-user developers of physical computing

artefacts. To my knowledge, this is the first tool to provide end-user developers with troubleshooting support via the medium of physical cards, in physical computing or any other domain.

The general aim of the card deck is to provide novice end-user developers with a wide range of tactics that can be used to improve diagnosis of physical computing problems, fixing of bugs, and evaluation of fixes, and to facilitate thinking/reflection during this process. The goal is not to give exhaustive and prescriptive check lists of instructions, but rather to encourage a creative and exploratory approach to troubleshooting, presented more as *scaffolding* than instruction.

Different approaches have been taken to provide end-user developer/programmers with support or scaffolding when debugging software programs. Most have adopted a technology-based approach, often relying on some form of background analysis of the program that is being debugged. More recently, and often with reference to the findings from the first study in this thesis (Booth et al. 2016), tools have begun to emerge to support the diagnosis of circuit bugs in physical computing prototypes, again, relying on background analysis what an end-user developer has built. To date, I have found only one tool which supports both programming and electronics—Bifröst (McGrath et al. 2017)—again, technology-based. My work to support end-user developers has taken a very different approach. Like most of these tools, and in accordance with Minimalist Theory principles (Carroll and Rosson 1987), it provides support *during the process* of debugging/troubleshooting. It aims to help end-user developers to solve their own problems (Cao et al. 2015), encourages hypothesis generation (Ko and Myers 2008), instantiates tactics or strategies (Cao et al. 2015) including planning and tracking of actions (Grigoreanu, Burnett, and Robertson 2010), and supports troubleshooting of circuit bugs (Drew et al. 2016; Wu, Shen, et al. 2017; McGrath et al. 2017), however it does all of these things through the medium of *physical cards*, which have been shown to have benefit for non-experts within a process, including ideation (Mora, Gianni, and Divitini 2017), the instantiation of knowledge (Bekker and Antle 2011) and the encouragement of reflection (Friedman and Hendry 2012).

The tool was inspired by popular creativity-support card decks, and content was informed by, firstly, empirical work which identified the most significant problems that end-user developers encounter when developing a physical computing device and analysis of their natural troubleshooting behaviours when dealing with these, and secondly, a review of the academic and non-academic literature on software debugging, hardware troubleshooting, physical computing and general problem solving. The design of the card deck was informed by a design

review of the academic literature on card-based tools, identifying key considerations when designing card-based tools, as well as focus groups with novice end-user developers and an informal pilot study, of an early prototype, with end-user developers. This process resulted in a card deck comprising 36 troubleshooting tactics cards, in five categories, as well as four component cards, and an additional category containing 6 best practice cards. The tactics encourage thinking, reflection and independent problem solving through the use of open questions. Other elements of the ‘Tactical Troubleshooting’ toolkit include a playmat encouraging hypothesis generation/prioritisation through selection of tactics, and a stand in which the cards can be stored/displayed.

Evaluation of this support tool delivers the next contribution of this thesis, which I will now describe.

7.1.4 Contribution 4

Insights into how troubleshooting support in the form of physical cards might be used and received by novice end-user developers.

The fourth contribution of this thesis resulted from addressing the following question:

TRQ4: What role might a card-based tool play in supporting end-user developers in the process of troubleshooting circuit bugs in a physical computing prototype? (Chapter 6, Study 2)

This contribution was achieved through an empirical, within-subjects user study in which twenty adult, novice end-user developers undertook troubleshooting tasks with and without the novel, card-based support tool.

While analysis of participants’ task success and bug-fixing performance did not find the tool to have conclusive positive effect on the outcome of end-user developers’ troubleshooting, the results suggest that it may decrease the likelihood of them being completely stuck.

Feedback from participants, via a questionnaire and interview, provides several insights into the role that a physical card-based support tool might play in a troubleshooting process. I discovered that support in form of physical cards may be well-received by some end-user developers, but not all. Several participants in this study liked the card format, and saw it as being less open to distraction than accessing support via a screen. Feedback suggests that

instantiating support as small chunks of information can make it easy to digest—a fun and informal way to access useful information, compared to some other types of support media, for example, lengthy texts and guides. The cards also show potential for accommodating different information-processing styles and reading abilities, allowing end-user developers to engage with as much or as little information as they wish. Participants in this study confirmed that troubleshooting cards afford benefits seen in previous cards research, including the creation of meaning or value through spatial arrangement—for example, grouping cards can be a convenient way to structure and track troubleshooting.

A number of effects were observed via feedback, broadly meeting the aims of the support tool.

Firstly, the study suggests that tactics cards can succeed in prompting ideas for different avenues of problem exploration and providing readymade suggestions for end-user developers struggling to formulate hypotheses for potential problem causes. While some participants in this study would have preferred more signposting in navigating the card deck and selecting tactics, drawing a random card can help to get end-user developers ‘unstuck’, leading them to areas of investigation that they have not considered, resulting in fault localisation. Troubleshooting tactic cards can also remind end-user developers of their existing troubleshooting knowledge—feedback suggests that that this not only has practical benefit in the moment, but may also increase end-user developers’ feelings of confidence and self-efficacy in their troubleshooting abilities. Some participants in the study even found the cards comforting—the wide range of tactics may reassure end-user developers that they still have plenty of troubleshooting options available. This perception of the cards as emotional support was unforeseen but encouraging. Feedback also suggests that support in this format may also have a positive priming or carryover effect on troubleshooting— from using the cards in a previous task or merely scanning or frontloading the tactics and best practice prior to troubleshooting. The cards may also serve as a good way to refresh degraded knowledge after a lengthy period of not having used Arduino—a common problem for end-user developers, experienced by several participants in this study.

Secondly, feedback indicates that support in this format can indeed succeed in encouraging end-user developers to think or reflect more—the current tool does this through the use of open questions, encouragement to shortlist tactics, and the ‘Stop... think’ category of cards, which particularly resonated with some as a reminder to take a step back from *doing*, to *reflect*. Some end-user developers find open questions helpful in guiding their thinking and encouraging independent problem solving, however this may not suit everyone. Like a few participants in this

study, some end-user developers may prefer concrete advice, need additional guidance when they struggle to answer open questions, or want additional information about components—the Component cards omitted from this study represent one way to address the latter. Equally, not all participants in this study were comfortable with the metacognitive activity of ‘thinking about their thinking’—a few felt it distracted them from ‘actual’ troubleshooting, while two participants found it cognitively overwhelming—effects perhaps exacerbated by the study constraints, but still worth noting.

Finally, devices such as the playmat in the current toolkit can provide useful functions within a troubleshooting process, but again this may not suit all end-user developers. Enforced use of the support materials did affect participants’ troubleshooting process, but participants’ opinions were polarised regarding this. Some found the playmat and shortlisting very useful, as a way to focus their thinking, and plan/structure their troubleshooting, helping them to stay on track and avoid some of their usual pitfalls—several observed thinking and behaving less chaotically than they did without the support materials, and making fewer speculative changes. Others, however, found the playmat a hindrance and the act of shortlisting unnecessarily time-consuming—having to think upfront conflicted with their usual approach, for example, trial-and-error tinkering. While these participants were resistant to effect on their troubleshooting, evidence of transformative reflection was observed in others—using the support materials made some aware of shortcomings in their existing process, which they now felt needed to change.

From the feedback reported, it seems that support in the form of physical cards shows potential for scaffolding novice end-user developers’ troubleshooting, providing a number of benefits. While this approach to support provision will not suit all end-user developers, it was well received by some and showed evidence of having met its aims, contributing a novel way to support novice end-user developers in their troubleshooting tasks.

7.2 Limitations of the work

The research undertaken in this thesis provides the first evidence of problems faced by end-user developers in the domain of physical computing development. I achieved this through studies involving representative users, i.e., end-user developers. As with any user study, limitations apply.

Each study involved a relatively small sample (20 participants), and participation was limited to those who were able to attend a 2-hour long in-person session in central London, at City, University of London's Interaction Lab. While London has a large maker community (e.g., the main hackerspace had well over 1,000 members at the time of recruitment), I nonetheless found it difficult to recruit participants, despite distributing my calls for participation widely and via several channels. This echoed my experience of recruiting for a previous study involving Arduino novices (Booth and Stumpf 2013). Recruitment for Study 2, focusing only on novices, proved even more difficult, as the eligibility criteria were even narrower.

One possible explanation for this may be low self-efficacy or lack of confidence within the target population. In the course of my associations with makers, before and during my PhD, I have become aware that confidence can sometimes be an issue, particularly for novices. This is unsurprising—for many people, confidence and feelings of self-efficacy grow with experience and knowledge. However, this has implications for any researcher wishing to recruit non-experts. While I did eventually meet my sample quota for each study, I must assume that my sample included only participants who felt confident enough to apply to participate, which means, they cannot be representative of all non-experts in this domain.

When I recruited participants for my first study, not much was known about this population. I cast the net widely, and resultingly, my sample included end-user developers of varying background and skill. As I have discussed, end-user developers are a diverse population with many motivations for engaging in physical computing development, and varying competencies. Whether different subsections of this population think or behave differently during development is unknown—my small sample size would not allow me to make reliable predictions of this kind. I am confident in my finding that end-user developers of all backgrounds experience problems, however, more work is required to determine whether particular types of experience unequivocally affect end-user developers' performance and behaviour in both development and troubleshooting of physical computing artefacts, beyond the analyses I do report.

The user studies I conducted took place under tightly controlled conditions, which may have affected the results—in section 7.4.1 I reflect further on this approach. I did not study end-user developers in their natural habitats, but in a laboratory setting, with strict time constraints, and the tasks were chosen by me, rather than by the participants themselves. End-user developers' behaviours may be different when working in more natural settings, upon projects of their own choice, without any time limitations, and not under scrutiny by a third-party. Participants in both

studies reported nervous feeling under pressure—pressure to succeed in the tasks, as well as time pressure. Although the tasks were relatively simple, as problem-solving tasks, they were still challenging for the end-user developers in my studies, as the results clearly demonstrate, and several participants showed frustration when they were unable to make progress or succeed in completing them. As I discuss in the previous chapter, frustrations in the final study were exacerbated by the additional requirement to use the support tool, again within specific constraints—a further cognitive challenge of using an unfamiliar tool on top of the challenge of troubleshooting in an unfamiliar system.

Think aloud may also have affected both participants' thinking and behaviour in these studies—I reflect further upon this in section 7.4.2. Several participants found it difficult to think aloud, and required regular reminders to do so. I also observed that when participants were experiencing a lot of difficulty, or became highly engrossed in problem solving, they required more reminders. We therefore cannot assume that the verbal protocol is a complete representation of participants thinking during the tasks. Pragmatically, therefore, much of my analysis in studies 1A and 1B involved not just what people *said* but my observations of what they *did*.

In the final study (Chapter 6), several participants performed better in Task 2 than in Task 1 (see section 6.3.1). While some of the improvement might be attributed to the support tool, evidenced by participants' comments regarding a positive/priming effect from using the cards (see section 6.3.2.2), other factors should be considered. Task 1 was participants' first exposure to the Love-O-Meter prototype, also, as they had been instructed not to refresh their knowledge prior to the session, for several participants, Task 1 also involved familiarising themselves with the platform. Additionally, while both groups were exposed to the additional challenge of having to use the support materials for one of their tasks, the WSNS group faced this in Task 1, on top of all of the challenges mentioned above. It seems reasonable to assume that performance in Task 1 and/or Task 2 may have been affected, to some degree, by any or all of these factors for some or all participants.

It is also difficult to isolate participants' opinion of the *cards* from their overall experience of the *support materials* in their entirety. While, ultimately, I am more interested in end-user developers' experience of the *cards* as a support mechanism, as I have described, the interview feedback confirms that the circumstances and constraints under which participants used the cards affected their opinions of them. Equally, many participants did not like the playmat. Many of the questions in the Support Materials Questionnaire referred to the '*Support Materials*' rather

than the cards, making it impossible to determine exactly what participants were rating—with hindsight, more specific wording could have been used. If participants—reasonably—interpreted these questions as referring to the support materials as a whole, rather than the cards alone, or participants’ ratings were weighted more towards particular elements of the support materials, this may have affected those ratings.

While the cards have shown some promise as a way to support end-user developers’ troubleshooting in this particular task, we can only speculate as to whether similar effects would be observed in a more complicated task, or one involving a different physical computing platform. More work would be needed to determine this.

Although I originally intended to look at end-user developers’ mental models of physical computing problems, the focus of this PhD changed in response to the initial findings—section 7.3 discusses this decision in more detail. How complete end-user developers’ mental models are in this domain, and what impact this has on their performance and behaviour in developing and troubleshooting physical computing prototypes, remain open questions.

Finally, I chose to use Arduino in my study, as it is currently the most well-known and widely used physical computing development tool, however, it is not the only tool available. Further work would be required to determine how generalisable the findings from my studies are to physical computing development involving other platforms.

7.3 The focus on troubleshooting

At a certain point in my PhD, I decided to focus the remainder of my research upon end-user developers’ *troubleshooting*. When I planned my first study, it was not only with the intention of investigating end-user developers’ difficulties—problems—in physical computing development, but also the mental models they held of physical computing concepts. This was inspired by the literature reporting circuit theory and fundamental programming concepts to be problematic for learners (discussed in sections 2.2 and 2.3.1, respectively). However, once data had been gathered, and analysis began to shape the findings about the problems experienced by participants during the hands-on development task, it became obvious that some patterns of behaviour in dealing with some of these problems might benefit from further investigation. For

example, some participants appeared to spend a great deal of time engaged in unproductive troubleshooting behaviours, and certain problems seemed particularly difficult to localise, and more prone to misdiagnosis, leading to more bugs.

The subsequent decision to prioritise looking deeper into troubleshooting behaviours, instead of analysing the interview data I had collected for mental models analysis, was therefore a pragmatic response to my initial research findings. It was also informed by a number of studies of end-user programmers' debugging, and how findings from that work, particularly in respect to strategies—patterns of debugging behaviour (section 2.2.3)—had proven useful in designing effective support tools for end-user programmers (section 2.8.1). Given what I was seeing in the study data I had gathered, investigating troubleshooting behaviours seemed a very logical approach for the next stage of my work. It also enabled me to extend our knowledge of end-user developers' problem-solving behaviours into the domain of physical computing development.

It is apparent in the findings from Study 1A and 1B that shortfalls in end-user developers' knowledge lies at the root of many of the problems they experience in physical computing development, when constructing and programming device prototypes, but also when troubleshooting. There are clear opportunities for support, however, as discussed earlier in the thesis (section 2.5.1), addressing end-user developers' knowledge gaps is more about situating useful information within their tasks, than providing the right education.

In choosing what tack to take in developing support for this population, I was particularly inspired by Jill Cao's Idea Garden approach to supporting end-user programmers (Cao 2013), that is, to help them to become better problem solvers, by gently suggesting approaches they might take, rather than solving their problems for them.

The decision to focus support on the *process* of troubleshooting, and on suggesting potentially useful tactics to use within it, was also very much a pragmatic choice. Dealing with problems is an inevitable part of any kind of development—even expert programmers and engineers spend a significant portion of their time debugging or troubleshooting.

There is consensus in the literature (e.g., Gick 1986; Perkins and Martin 1986; Jonassen 2010) that troubleshooting and debugging require several different types of knowledge, both domain-specific and generic, and that efficacy grows with experience. Knowledge gained through exposure to different problems enables experienced troubleshooters/debuggers to quickly determine strategies or avenues of inquiry most likely to aid in diagnosis of a problem, based on

their understanding of the system and historical, problem-related data they have assimilated. This puts less-experienced end-user developers, including novices, at an obvious disadvantage.

Suggesting troubleshooting tactics—of varying specificity—that end-user developers might use, addresses a particular knowledge gap—a deficit of *troubleshooting* knowledge—however, I am conscious that this represents only one piece of the puzzle, and is certainly no panacea for all end-user developers’ knowledge shortfall-related problems in this domain. Support addressing *different* knowledge gaps might also be effective in helping end-user developers to troubleshoot, or even avoid some problems, however, it is worth noting that more domain knowledge did not *always* equate with better performance in the studies I conducted—a professional engineer (P02) in Study 1A/1B (see section 4.4.1) still introduced errors in circuit construction which led them to fail the task.

Providing end-user developers with different options to consider when troubleshooting has the potential not only to aid in diagnosing any immediate problem, but also problems later down the line, including when using different tools or platforms—many of the tactics are generic or flexible enough to be applied to many different situations or contexts. The approach I have taken is, I believe, a good first step in equipping end-user developers with tools to help them tackle—and hopefully overcome—the problems that they will inevitably encounter during development, irrespective of their physical computing expertise.

7.4 Reflection on methods and approach

Completing the thesis—and with it, my PhD journey—provides an opportunity to take stock, and reflect on the research I have undertaken during the course of it, and what I have learned beyond the research results and findings reported.

In section 1.5 I refer to the methodological stance underpinning my work, and the impact this has had upon my general approach to research and my choice of research methods. I return to some of these choices with a critical eye.

7.4.1 Task observation in a laboratory environment, under tight constraints

First-hand observation or measurement of representative users undertaking tasks is the de facto method within user research for uncovering problems in the use of technology—conventional wisdom being that findings based on observable, measurable data are more reliable than the subjective opinions of participants, the latter being subject to memory/attention constraints and fallibilities, but also a number of potential biases (Nielsen 2001).

Two major rounds of data collection are described in this thesis. Both involved observing end-user developers undertaking given tasks in a laboratory environment, subject to a number of constraints, including strict time limits and a requirement to think aloud, while being video recorded, but also a requirement to use materials—most specifically the support tool—in a way that did not match their usual behaviours.

Both samples included *novice* end-user developers—some participants in the first study; all in the final study—and tasks which, based on observed performance, were challenging for most. For novices in these studies, participation meant doing something at which they were not very skilled, or knowledgeable about—under scrutiny, with video cameras recording every aspect. As a responsible researcher, I did everything possible to ensure that participants were treated both ethically and well throughout their involvement, and took pains to try to put them at ease in the laboratory sessions, however, I was conscious of the pressure clearly felt by some as a consequence of the nature and circumstances of the study—at various points during the tasks, in both studies, participants' nerves, frustrations and/or performance anxieties were apparent, whether in their verbal comments or body language.

The potential effect upon my study results has been acknowledged in the Limitations section of this chapter (section 7.2), and I by no means feel that these factors invalidate the work I have done, nor the findings that I report. However, it does make me pause to consider whether alternative study designs or different methods might avoid or mitigate such an effect while delivering at least equal, if not better, value. This remains an open question.

7.4.2 Problems with think aloud

Think aloud aims to expose *what is in a participant's head* to the researcher—data that would not otherwise be available—through verbalisation. All of my studies used this method.

Over the course of my research, I watched forty-four end-user developers spend a collective total of almost 35 hours problem solving in physical computing tasks while thinking aloud. What I observed is that while some end-user developers seem naturally capable of almost a verbal stream of consciousness while undertaking a task—*“I think I talk like this anyway, even when no one is around”*—far more find it challenging to think aloud effectively when engrossed in problem solving, and require frequent prompts or reminders to do so. It also seems that some participants are more inclined to provide very cursory descriptions of what they are doing—*‘I am removing the wire’*—rather than relaying their thoughts. The resulting inconsistency of the verbal protocol meant that I was forced to rely more on behavioural data—i.e., participants’ actions—than originally planned for analysis in Study 1A and 1B. While I believe this to have been effective in answering the research questions for these studies, my takeaway from this experience is that the quality of a verbal protocol is highly dependent, at least in part, on how ‘good’ participants are at think aloud—at verbalising their thoughts in parallel with performing the task itself, particularly if the task is difficult or requires great concentration—and how comfortable they are with speaking their thoughts aloud under scrutiny.

Others have reported problems with think aloud during debugging (e.g., Fitzgerald et al. 2008). Threats to the validity of think aloud data have been raised within the literature (e.g., Russo, Johnson, and Stephens 1989), and despite being very commonly used in HCI studies there is inconsistency of opinion regarding how best the method should be applied (Boren and Ramey 2000). I am led to reflect upon whether applying this method in a different way, or employing a different method altogether, in future studies, might offer some improvement, for example, although recruiting *pairs* of participants may be more challenging, having two participants talking to one another as they work together has been shown to yield more information than one person thinking aloud (Dumas and Redish 1999, 31).

7.4.3 Approach to analysis

Although all of my empirical studies involved a mixed methods approach, there was a considerable difference between the first two studies (study 1A and 1B) in comparison to the final one (study 2).

Data analysis in studies 1A and 1B focused, primarily, on observable evidence of participants' problems during development, and their behaviours in tackling them. Task data were carefully transcribed, unitised, and categorised, using coding schemes informed by the literature, then transformed into numerical data in order to subject them to quantitative methods, providing measurements and comparisons of performance and behaviour. Verbatim quotes from the think aloud are used to illustrate particular findings, however participants' reflections upon their performance in the task, or the difficulties that they experienced, are mostly absent.

In contrast, although study 2 provides some quantitative measures of performance and opinion, qualitative analysis now focused on the post-tasks debriefing interview, using a thematic analysis approach. This aimed to elicit participants' subjective opinions of the support tool, based on their first-hand experience of using it. Rather than seeking to categorise and quantify the qualitative data captured, I teased themes from it that not only yielded feedback about the tool, but also exposed some of the factors that potentially affected participants' performance in the tasks and coloured their impression of the tool, whether positively or negatively. Giving voice to participants in a way that the previous studies had not was crucial to understanding the data that had been captured, providing an additional layer of context to participants' behaviour during the tasks, their task performance results and their response to, and opinions of, the support tool. It also prompted me to reflect, once again, on methods that I take for granted as being 'what works' for a particular type of problem.

7.4.4 'What works' vs 'What might work better?'

While I would like to be able to provide unequivocal proof of the effects of the physical cards-based support tool upon participants' troubleshooting in Study 2, participants' feedback suggests that the study design was not ideal for this. I do believe that the findings reported provide more than enough evidence that instantiating troubleshooting process support in the form of physical cards does hold promise as a novel way to help end-user developers overcome their difficulties. However, the value of the study, in terms of my own learning, extends beyond

the answers to the research questions, by highlighting that a typical, controlled ‘debugging study’ design might not be the best way to evaluate this type of tool in this domain, and that any future evaluation of this particular tool might benefit from a different approach. This might include, for example, more *naturalistic* studies that are more in alignment with the ethos of making as a creative and pleasurable activity, discussed earlier in the thesis (section 2.1.3).

7.5 Opportunities for future work

The limitations I have described, as well as the findings from my studies, and my reflections upon the approach and methods I have used, highlight opportunities for future work.

As I have shown, many of the participants in the final study liked the cards and found them useful, however the circumstances under which they were required to use them led to some frustration. More work is needed to determine the efficacy of this tool, including analysis of the behaviours it engenders and the benefits it may have, for example, improvements in troubleshooting skill, including the choice of tactics, and whether learning transfers to further/future tasks. Additionally, studies in a more naturalistic, or less constrained setting could provide a better idea of how the tool would naturally be used by end-user developers outside of a controlled environment.

Some participants had trouble working out how to use the cards and would have liked more guidance, for example, in where to start and in selecting tactics to use. A future iteration of the toolkit could explore the potential for a more structured method, or signposting end-user developers towards content most appropriate for the problem they are experiencing. Similarly, it would be interesting to explore a participant’s suggestion about indicating tactic specificity, enabling end-user developers to filter tactics according to how general or specific an approach they wanted or needed to take at a particular juncture. This may even have benefit beyond the use of one particular tool or in this domain, in terms of increasing end-user developers’ general troubleshooting knowledge. More work would be needed to determine the best way to implement these suggestions.

While, based on interview feedback in the final study, the cards succeeded in making participants think or reflect more when troubleshooting—one of the primary aims of the support

tool—participants suggested that additional information would be useful, for example, concrete examples of how to connect certain components, or pointers to additional guidance, such as component data sheets. There are, of course, the Component cards (section 5.5.4), which were excluded from the evaluation study, to ensure focus on the troubleshooting tactics, however this type of information could also be provided in different ways, including digitally, linked to from the physical cards (for example, simply through a QR code), directly within a mobile application equivalent, or in some other form.

As participants' comments suggest, the support tool in its current form will not suit everyone. The information it provided was generally seen as very useful, but while many appreciated the physical card-based format, some would have preferred it in a different form, for example, a software application, as was suggested by more than one participant. Some of the card tools reviewed in the literature have digital equivalents, for example, the Game Design Deck of Lenses (Arcila 2013), exists both as a physical card deck and a mobile phone app, as does the Oblique Strategies deck ('The Oblique Strategies', n.d.). We could do similar with this tool. While some of the flexibility of the cards would be lost in a digital form, for example, the ability to review, or work with, a number of cards at once, or use spatial arrangements for planning and tracking, there may be other benefits, for example, in the ability to filter or search for tactics by specific criteria, or direct links to additional content.

However, more innovative hybrid physical/digital approaches could be explored, combining the benefits of tangible cards, with the flexibility of a digital medium in respect to dynamic provision of information. For example, the use of an interactive playmat TUI (tangible user interface) in conjunction with cards printed using conductive ink, or embedded with NFC chips. Augmented/mixed reality technologies could also be used to extend information beyond the boundaries of physical cards.

It would also be interesting to see whether the tool has benefit in other contexts of use, beyond use by individual adult end-user developers. I have already received interest in using the tool to support and teach troubleshooting in education, for example, within school, adult education or university classroom settings. One participant in the final study, a freelance educator, would like to adapt the cards for use in the Scratch programming classes they teach at primary schools, where they see them being used in pair-based troubleshooting.

Finally, I intend to refine the cards in light of the results of the evaluation study, and make the toolkit available for download, extension, and customisation. I see toolkits such as this as a vital step towards greater adoption and continued use of physical computing technology by novice end-user developers.

Appendices

Appendix A. Study 1A Ethics application

Ethics Proportionate Review Application: Staff and Research Students Computer Science Research Ethics Committee (CSREC)

Staff and research students in the Department of Computer Science undertaking research that involves human participation must apply for ethical review and approval before the research can commence. If the research is low-risk, an application can be submitted for a proportionate review using this form. Applicants are advised to read the information in the SMCSE Framework for Delegated Authority for Research Ethics prior to submitting an application.

There are two parts:

Part A: Ethics Checklist. The checklist determines whether the research is low-risk. If it is, Part B of the form should also be completed. If not, the checklist provides guidance as to where approval should be sought, but the checklist itself does not need to be submitted.

Part B: Ethics Proportionate Review Form. This part is the application for ethical approval of low-risk research and should only be completed if the answer to all questions (1 – 18) is NO.

Completed forms should be returned to the Chair of CSREC by email (*address redacted*).

Part A: Ethics Checklist

If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval:		
1.	Does your research require approval from the National Research Ethics Service (NRES)? (E.g. because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/)	No
2.	Will you recruit any participants who fall under the auspices of the Mental Capacity Act? (Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/)	No
3.	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? (Such research needs to be authorised by the ethics approval system of the National Offender Management Service.)	No
If your answer to any of the following questions (4 – 11) is YES, you must apply to the Senate Research Ethics Committee for approval (unless you are applying to an external ethics committee):		
4.	Does your research involve participants who are unable to give informed consent, for example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	No
5.	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	No
6.	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	No
7.	Does your research involve participants disclosing information about sensitive subjects?	No
8.	Does your research involve the researcher travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (http://www.fco.gov.uk/en/)	No
9.	Does your research involve invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	No
10.	Does your research involve animals?	No
11.	Does your research involve the administration of drugs, placebos or other substances to study participants?	No

If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		
12.	Does your research involve participants who are under the age of 18?	No
13.	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	No
14.	Does your research involve participants who are recruited because they are staff or students of City University London? For example, students studying on a particular course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	No
15.	Does your research involve intentional deception of participants?	No
16.	Does your research involve participants taking part without their informed consent?	No
17.	Does your research pose a risk to participants greater than that in normal working life?	No
18.	Does your research pose a risk to you, the researcher(s), greater than that in normal working life?	No
You must make a proportionate review application to the CSREC if your research involves human participation and you are not submitting any other ethics application (i.e. your answer to all questions 1 – 18 is "NO").		

Part B: Ethics Proportionate Review Form

If you answered NO to all questions 1 – 18, you may use this part of the form to submit an application for a proportionate ethics review of your research. The form must be accompanied by all relevant information sheets, consent forms and interview/questionnaire schedules.

Note that all research participants should be fully informed about: the purpose of the research; the procedures affecting them or affecting any information collected about them, including information about what they will be asked to do, what data will be collected, how the data will be used, to whom it will be disclosed, and how long it will be kept; the fact that they can withdraw at any time without penalty.

Background Information	
Name:	Tracey Booth
Supervisor (if student):	Dr Simone Stumpf
Your Research Project	
Title:	Exploring How End-user Developers Think and Behave When Developing Physical Prototypes
Start date:	13/03/2015
End date:	30/04/2016
The current 'Maker Movement' entices end users into constructing and programming microcontroller-based prototypes for personal use, however, not much is know yet about this growing subgroup of end-user developers (EUDs). Poor or erroneous mental models affect students' learning and application of circuit theory, and are a significant source of novice programmers' difficulties. Additionally, 6 learning barriers have been identified, which can stall end-user programmers' progress. Knowing how both apply in a physical prototyping context will help us understand how technology can support EUDs. My main research questions are: RQ1. What learning barriers do EUDs encounter when constructing and programming physical prototypes? RQ2. What mental models of physical prototyping concepts do EUDs hold? RQ3. Are there common incorrect mental models that impact EUDs' physical prototyping progress and success? Participants will be 20 adults, of varying background and ability, who use the Arduino platform to develop physical prototypes for personal use. I will recruit via hackerspaces and other 'Maker' community groups. People who respond to the initial email will be contacted and screened, including for disability, to ensure that they meet the criteria for participation and are capable of performing the required activities. Participants will be sent an online questionnaire, gathering data about their background and experience in programming, electronics and physical prototyping. At the start of this questionnaire they are asked to provide consent to their response data being stored and used for the purpose described. Once they have completed the questionnaire, each participant will attend an hour-and-a-half-long session in City's usability lab, at a time	

convenient for them. This session is structured into 3 phases:

1. A self-efficacy questionnaire to measure their confidence in physical prototyping;
2. A hands-on task, in which they develop a physical prototype using the Arduino platform, a solderless breadboard and a kit of components, with access to help resources. A verbal protocol (think aloud) will be used, and both on and off-screen actions video recorded;
3. A semi-structured interview, in which they explain the prototype workings, and answer questions (selected from a guiding list of topics as time allows) to elicit their mental models of the concepts involved. I will also probe on issues observed, including misconceptions or areas of difficulty.

The session sequence takes into account the potential for each activity to affect data gathered in subsequent phases. Participants will be asked to sign an Informed Consent form at the start of the session, before any data is gathered.

Once gathered, any identifying data will be anonymised - participants will be represented by randomly assigned ID numbers. Participant names will not be associated with the recordings or any other data, and will not appear in any reports or presentations, including where any video clips or screenshots are used in which faces are shown. All data will be password protected, stored securely, and backed up. Only myself, my supervisors (Dr Simone Stumpf, Dr Sara Jones and Dr Jon Bird, and my external examiners, will have access to the data. If a participant decides to withdraw from the study at any point, I will destroy any data already gathered from them.

Analysis will involve mixed methods. RQ1 will be addressed by coding recording transcripts for learning barriers - I will look for types and frequencies of barriers encountered. To answer RQ2 I will perform a thematic analysis of the mental models elicitation data to determine the mental models held by EUDs of physical prototyping concepts. To answer RQ3 I will identify common mental model types, misconceptions and knowledge gaps in this thematic and investigate whether these are correlated with physical prototyping performance and efficacy.

In addition, I have a number of secondary research questions, as follows:

- RQ4. Is there a relationship between EUD's backgrounds and their mental models of physical prototyping concepts, including any misconceptions they hold?
- RQ5. Is there a relationship between EUDs' backgrounds and the learning barriers they experience when constructing and programming physical prototypes?
- RQ6. Is there a relationship between EUDs' self-efficacy and any other factor?

I will therefore look for correlations between participants' backgrounds, their self-efficacy scores, their mental models and the learning barriers they experience. The task recordings will also allow me to analyse participants' strategies for prototype development, such as whether and how they seek out and use existing examples or instructions, and their behaviour and efficacy in the use of help content.

Attachments (these must be provided if applicable):	
Participant information sheet(s)	Yes
Consent form(s)	Yes
Questionnaire(s)	Yes
Topic guide(s) for interviews and focus groups	Yes
Permission from external organisations (e.g. for recruitment of participants)	Not applicable

Appendix B. Study 1A Recruitment poster

Arduino users needed

for a study at City University London



Do you use Arduino for personal projects or to support/produce your own work?

I'm looking for people to take part in an exploratory study I'm conducting as part of my PhD.

You don't need to be an expert to take part, just to have **experience of using the Arduino platform**, and be willing to attend an in-person session at City University London, in which you do a hands-on task (using Arduino) and answer some questions about it.

The session will take about 1.5 hours and you'll receive a £20 Amazon gift voucher as a token of thanks. (There will also be chocolate involved)

Interested? Drop me a line: tracey.booth.1@city.ac.uk

Tracey Booth
PhD research student | Centre for Human-Computer Interaction Design
School of Maths, Computer Science & Engineering | City University London

See also <http://tinyurl.com/arduinostudy>



CITY UNIVERSITY
LONDON



Participant Information Sheet

Title of study: Exploring How End-user Developers Think and Behave When Developing Physical Prototypes

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it's important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. Ask us if there is anything that is not clear or if you would like more information.

What is the purpose of the study?

I'm a PhD research student in the Centre for Human-Computer Interaction Design (HCID) at City University London, and a keen Maker. The main aim of my research is to determine how people can be supported in learning and using physical prototyping platforms such as Arduino. As you are probably aware, the current 'Maker Movement' is leading more and more people from different backgrounds to have a go with Arduino and similar platforms, but not all are experts in electronics and programming. Through this exploratory study, which starts in April 2015, I'm hoping to get a broad picture of how different individuals think and behave when creating and programming physical prototypes.

Why have I been invited?

20 adult Makers, who use the Arduino platform to create physical prototypes for personal use, are being invited to take part in this study. You don't need to be an expert. The physical prototyping community is diverse, so I need to involve people from different backgrounds, with different levels of knowledge and skill. You must have some experience of using the Arduino platform (hardware and official IDE), but must not have developed physical prototypes (creating circuits and programming their behaviour) as your main job function, or commissioned by other people. You must also be at least 18 years old and have no physical or cognitive disability that would prevent you from undertaking the activities described below.

Do I have to take part? What happens if I sign up and then change my mind?

Participation is voluntary - it is up to you to decide whether or not to take part. If you do decide to take part you will be asked to sign a consent form, but are still free to withdraw at any time and without giving a reason - you will not be penalised or disadvantaged in any way and any data we have collected from you will be destroyed.

What will happen if I take part?

You will first be sent a link to an online questionnaire for you to complete, which will ask about your background and experience in physical prototyping, electronics and programming. We will then schedule an individual session with me (Tracey Booth), in the Interaction Lab at City University London, to take place in April 2015 at a time convenient for you. The session will take approximately 1.5 hours and will comprise:

- 1) A short self-efficacy ratings questionnaire,
- 2) A hands-on physical prototyping task (breadboard circuit construction, and programming) using the Arduino platform (board + official IDE) to a given brief, and finally,
- 3) Some verbal questions about the prototype you develop.

The session will be video-recorded, including audio.

Participants who complete the study will be given a £20 Amazon gift voucher, as a small token of thanks for taking part.

Please note that this is an exploratory study, not an experiment or test. Data analysis will be mostly qualitative and the findings will be used to inform my next studies in this area.

Appendix C. Study 1A Participant information sheet

What are the possible benefits of taking part?

The goal of end-user development research is to empower end users to develop and adapt systems themselves. By taking part in this study you will contribute to our understanding of how different people think and behave when using platforms such as Arduino, and the types of challenges they face. Longer term this knowledge can be used to inform the design of tools and appropriate supports for people, particularly novice Makers. Thereby we hope this research will benefit not only individuals, but ultimately Maker communities in general, by making it easier for more people to get involved. We want to lower the barriers for people to get to grips with physical prototyping, and provide them with a better experience.

Will my taking part in the study be kept confidential?

Yes. Only myself, my PhD supervisors (Dr Simone Stumpf, Dr Sara Jones and Dr Jon Bird) and external examiners, will have access to the data gathered, however data will be anonymised once gathered, and stored securely. Your name will not be associated with the recordings or any other data, and will not appear in any reports or presentations.

What will happen to results of the research study?

This study will be written up to form part of my PhD thesis and the results will inform the next studies I undertake towards its completion. In addition, I hope to submit my findings for academic publication in international journals and conferences in the field of human-computer interaction.

What will happen when the research study stops?

Data will be stored for 10 years - the standard retention time for university studies - and then destroyed.

What if there is a problem?

If you have any problems, concerns or questions about this study, you should ask to speak to a member of the research team. If you remain unhappy and wish to complain formally, you can do this through the University complaints procedure. To complain about the study, you need to phone 020 7040 3040. You can then ask to speak to the Secretary to Senate Research Ethics Committee and inform them that the name of the project is: Exploring How End-user Developers Think and Behave When Developing Physical Prototypes

You could also write to the Secretary at:

Anna Rambert
Secretary to Senate Research Ethics Committee
Research Office, E214
City University London
Northampton Square
London, EC1V 0HB
Email: Anna.Rambert.1@city.ac.uk

City University London holds insurance policies which apply to this study. If you feel you have been harmed or injured by taking part in this study you may be eligible to claim compensation. This does not affect your legal rights to seek compensation. If you are harmed due to someone's negligence, then you may have grounds for legal action.

Who has reviewed the study?

This study has been approved by City University London Computer Science Research Ethics Committee (CSREC).

Further information and contact details

Researcher: Tracey Booth tracey.booth.1@city.ac.uk
PhD supervisor: Dr Simone Stumpf simone.stumpf.1@city.ac.uk +44 (0)20 7040 8168

Thank you for taking the time to read this information sheet.

Appendix D. Study 1A Informed consent form



**CITY UNIVERSITY
LONDON**

INFORMED CONSENT

Study: Exploring How End-user Developers Think and Behave When Developing Physical Prototypes

Please
initial box

1.	I agree to take part in the above City University London research project. I have had the project explained to me, and I have read the participant information sheet, which I may keep for my records. I understand this will involve: • Completing a questionnaire about my confidence in completing physical prototyping tasks. • Using provided equipment, including a computer, to develop a physical prototype. • Being interviewed by the researcher • Allowing the session to be video-recorded, with audio.	
2.	This information will be held and processed for the following purpose(s): • PhD research I understand that any information I provide is confidential, and that no information that could lead to the identification of any individual will be disclosed in any reports on the project, or to any other party. No identifiable personal data will be published. The identifiable data will not be shared with any other organisation. AND I consent to the use of sections of the video recordings in publications and/or presentations.	
3.	I understand that my participation is voluntary, that I can choose not to participate in part or all of the project, and that I can withdraw at any stage of the project without being penalised or disadvantaged in any way.	
4.	I agree to City University London recording and processing this information about me. I understand that this information will be used only for the purpose(s) set out in this statement and my consent is conditional on the University complying with its duties and obligations under the Data Protection Act 1998.	
5.	I agree to take part in the above study.	

Name of Participant

Signature

Date

Name of Researcher

Signature

Date

Page 2/6

Page 2: About you

- What is your full name?**
*First name & last name, e.g. Tracey Booth. This will be anonymised later.**

- 2) What is your age?
In years, e.g. 37*

- 3) What is your gender?
- ☐ Female
- ☐ Male
- ☐ ...

- What is your current occupation / profession?
E.g. PhD student (Computer Science); University Lecturer (Maths); Interaction Designer *

Next (navigates to Page 3)

Physical prototyping study - Background questionnaire

Page 1: Informed Consent

Hello and thank you for agreeing to take part in my research. This survey has 3 pages, and is structured as follows:

- Page 1: Informed consent (this page)
- Page 2: Your details (name, age, gender, occupation)
- Page 3: Your background and experience in physical prototyping, programming and electronics.

Please read the following statement and check the box to continue. If you have any problems with the survey, please contact me on 07##### or tracey.booth.1@city.ac.uk

Many thanks,

Tracey

Informed consent

The responses you provide in this survey will be used for the purpose of my PhD research at City University London. Only me (Tracey Booth), my PhD supervisors (Dr Simone Stumpf, Dr Sara Jones and Dr Jon Bird) and my external examiners will have access to the responses you provide, however the data will be anonymised, and stored securely. Your name will be replaced with an ID number and will not be associated with any data. Your name will not appear in any reports or presentations.

Please confirm below that you have read the above statement and consent to the data being used in the way described. (If you do not consent then please close this browser window to exit the survey and contact me to say that you do not wish to continue)*

- ☐ I have read and understand the Informed Consent statement, and consent to my responses being used as described.

Next (navigates to Page 2)

Page 3: Background & Experience

1. Physical prototyping

(Creating electronic circuits AND programming them, e.g. with the Arduino platform)

5) How long have you been developing physical prototypes? *

Time Years Months

6) How long have you been developing physical prototypes with the Arduino platform? *

Time Years Months

7) How expert do you think you are at physical prototyping (in general)? *

Complete beginner 1 2 3 4 5 6 7 Totally expert

8) How expert do you think you are at physical prototyping with the Arduino platform? *

Complete beginner 1 2 3 4 5 6 7 Totally expert

9) How would you say you acquired your current physical prototyping knowledge and skills, including Arduino? *

- ☐ Totally self-taught
- ☐ Mostly self-taught
- ☐ Equally self-taught and through training/instruction
- ☐ Mostly through training/instruction
- ☐ Totally through training/instruction

10) Have you had any training or instruction in developing physical prototypes with Arduino or any other physical prototyping platform? *

I have had training* Courses, workshops, modules, tutoring etc.

☐ Yes ☐ No

What training?*

An idea of what, where, and for how long would be useful. If you can't remember details, please just provide a summary.

2. Programming (general)

11) How long have you been programming? *

Time Years Months

12) How expert do you think you are at programming (in general)? *

Complete beginner 1 2 3 4 5 6 7 Totally expert

13) Have you ever been employed as a professional programmer? *

☐ Yes ☐ No

14) How would you say you acquired your current programming knowledge and skills? *

- ☐ Totally self-taught
- ☐ Mostly self-taught
- ☐ Equally self-taught and through training/instruction
- ☐ Mostly through training/instruction
- ☐ Totally through training/instruction

15) Have you had any training or instruction in programming? *

I have had training* Courses, workshops, modules, tutoring etc.

☐ Yes ☐ No

What training?*

An idea of what, where, and for how long would be useful. If you can't remember details, please just provide a summary.

3. Electronics (general)

16) How long have you been constructing/troubleshooting electronics circuits? *

Complete beginner

1

2

3

4

5

6

7

Totally expert

Time

Years

Months

17) How expert do you think you are at constructing/troubleshooting electronic circuits (in general)? *

18) Have you ever been employed as a professional electronics engineer, or specifically to construct/troubleshoot electronic circuits? *

Yes

No

19) How would you say you acquired your current electronics knowledge and skills? *

Totally self-taught

Mostly self-taught

Equally self-taught and through training/instruction

Mostly through training/instruction

Totally through training/instruction

20) Have you had any training in constructing/troubleshooting electronic circuits? *

I have had training* Courses, workshops, modules, tutoring etc.

Yes

No

What training?*

An idea of what, where, and for how long would be useful. If you can't remember details, please just provide a summary.

Finally...

21) Any other comments you would like to add about your background and/or experience?

Next (navigates to Thank you)

Thank You!

Thank you very much for completing the questionnaire.

Appendix F. Study 1A Self-efficacy questionnaire

Physical Prototyping with Arduino: Self-efficacy

What is your full name? (first name and last name, e.g. Tracy Booth)*

Please answer the following question for each of the statements below. If you answer Yes for a statement, please rate your confidence in completing the task.

"I could complete a physical prototyping task of moderate complexity using the Arduino platform..."

1) ... if there was no one around to tell me what to do as I go

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

2) ... if I had never used a platform like it before

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

3) ... if I had only the online reference materials for reference

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

4) ... if I had seen someone else using it before trying it myself

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

Thank You!

5) ... if I could call someone for help if I got stuck

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

6) ... if someone else had helped me get started

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

7) ... if I had a lot of time to complete the task for which the platform was provided

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

8) ... if I had only the built-in help facility for assistance

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

9) ... if someone showed me how to do it first

I could complete the task*

() Yes () No

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

10) ... if I had used similar platforms before this one to do the same task

I could complete the task*

() No () Yes

Confidence rating (if Yes)*

Not at all 1 2 3 4 5 6 7 8 9 10 Totally confident () () () () () () () () () ()

Appendix G. Study 1A Task instruction sheet

Hands-on physical prototyping task

Goal 
• Please construct and program an Arduino-based physical prototype that uses a sensor to read temperature, and at least 3 LEDs to indicate levels of warmth as temperature changes. ○ Use the sensor to read the temperature ○ As temperature increases, more LEDs should be lit. ○ As temperature decreases, fewer LEDs should be lit. ○ No LEDs should be lit for the ambient room temperature. • You can change the temperature that the sensor reads by holding it between your fingers.

Do 	Don't 
• Use the Arduino UNO board and IDE • Keep it simple • Remember to 'think aloud' • Use help/online resources, if needed, e.g. to look up commands, components	• Don't search online for an existing solution and copy it.

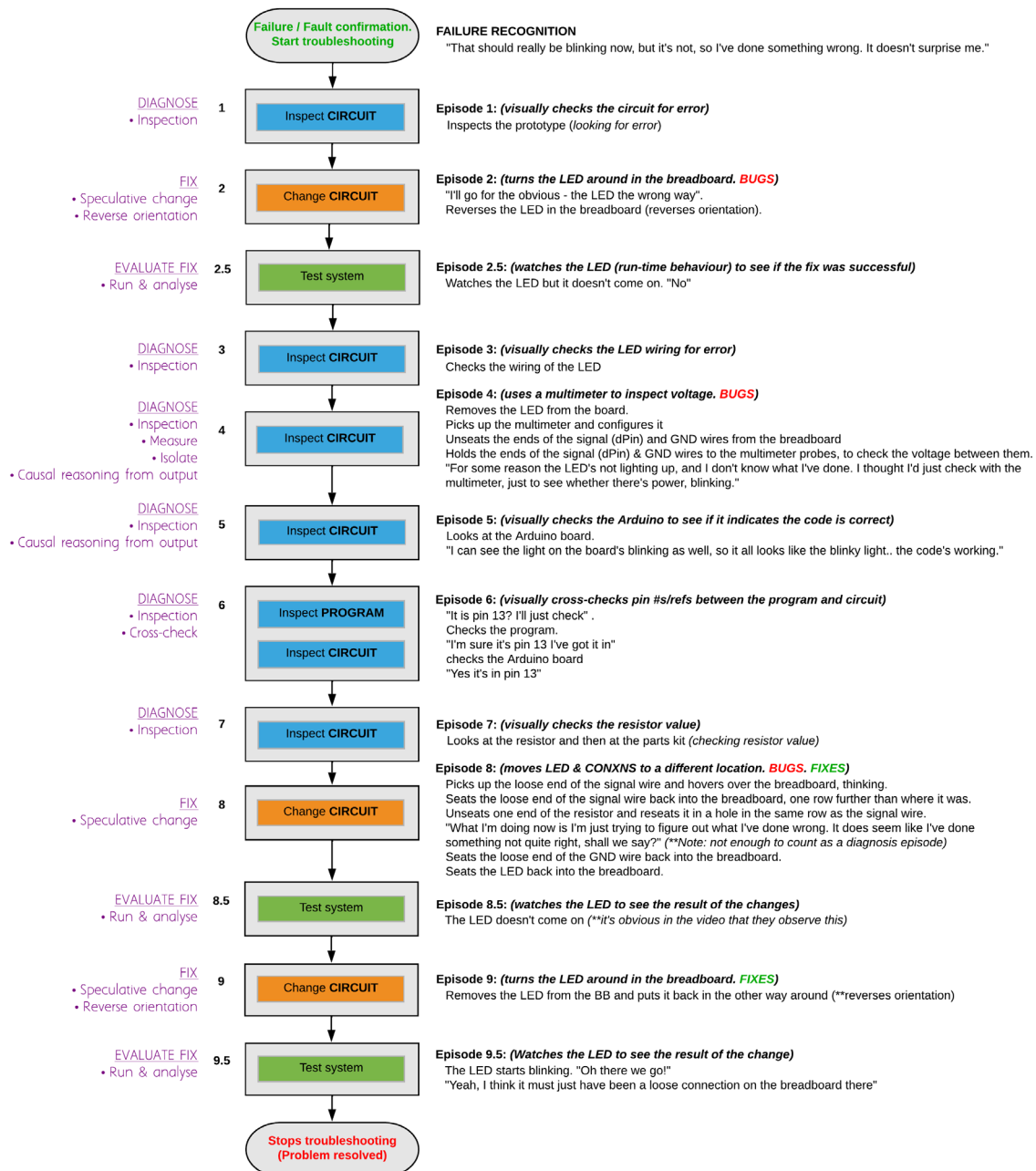
Time 
You have <u>up to 45 minutes</u> for this task.

Appendix H. Study 1B Troubleshooting flow diagram

A visual representation of the sequence/flow—and summarised content—of episodes within a run.

- Each sequentially numbered block following the *Failure recognition* oval represents an *episode*.
- To the right of each episode is a summary of what occurred in it, and the associated text from the transcript.
- The coloured blocks within each episode block summarise the *Event Types* coded. E.g., in episode 8, the participant made several changes to their circuit—the multiple *Change* event types coded in the transcript spreadsheet, each with the *C* subcode, are here represented by a single ‘*Change circuit*’ block.
- Event types were dual coded using words and colours in the transcript spreadsheets. The same colours were used in the troubleshooting flow diagrams, for visual clarity, e.g., *Change* blocks are always orange, etc.
- The diagrams were hand-coded with *Activity Type* and *Tactic* codes. Episodes in the image below are annotated with this coding, in purple—Activity Type codes are capitalised and underlined, with *Tactic(s)* beneath

Circuit bug: LED's resistor not seated properly in the breadboard (resolved)



Appendix I. Initial set of candidate tactics

A list of the tactics first compiled for potential inclusion in the troubleshooting support card deck. The tactics are grouped by their initial categories. Please note that this is not the final set of tactics or categories used in the final study. The final set is available as Appendix L.

Category title	Tactic
Check component / wiring info	Copy an example
	Get more help
Generate more data	Logging statements
	Measure something
	Use tools
Inspect for build errors / faults	Check circuit completeness
	Check for bad connections
	Check for special cases/uses
	Check location of failure
	Check order / sequence
	Check power
	Check seating
	Check the pinout
	Check the type
	Check the value
	Check wiring
	Compare to an example
Perform a test	Cross-check
	Change test input
	Check for faulty component
	Swap working & non-working
Simplify	Unit test
	Divide & conquer
	Isolate
Try a quick fix	Reduce dependencies
	Redo (the same way)
	Reseat
	Restart
	Reverse orientation
	Swap for a different value
	Swap for new identical
Understand (define) the problem	Check conditions
	Check for [ab]normality
	Check frequency / consistency
	Check order / sequence
	Check output values
	Reproduce the problem
Understand the system	Similar/familiar problem
	Check the brief
	Identify / trace dependencies

Appendix J. Card design focus groups ethics application

Ethics Proportionate Review Application: Staff and Research Students Computer Science Research Ethics Committee (CSREC)

Staff and research students in the Department of Computer Science undertaking research that involves human participation must apply for ethical review and approval before the research can commence. If the research is low-risk, an application can be submitted for a proportionate review using this form. Applicants are advised to read the information in the SMCSE Framework for Delegated Authority for Research Ethics prior to submitting an application.

There are two parts:

Part A: Ethics Checklist. The checklist determines whether the research is low-risk. If it is, Part B of the form should also be completed. If not, the checklist provides guidance as to where approval should be sought, but the checklist itself does not need to be submitted.

Part B: Ethics Proportionate Review Form. This part is the application for ethical approval of low-risk research and should only be completed if the answer to all questions (1 – 18) is NO.

Completed forms should be returned to the Chair of CSREC by email ([email address redacted](#)).

Part A: Ethics Checklist

If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval:		
1.	Does your research require approval from the National Research Ethics Service (NRES)? (E.g. because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/)	No
2.	Will you recruit any participants who fall under the auspices of the Mental Capacity Act? (Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/)	No
3.	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? (Such research needs to be authorised by the ethics approval system of the National Offender Management Service.)	No
If your answer to any of the following questions (4 – 11) is YES, you must apply to the Senate Research Ethics Committee for approval (unless you are applying to an external ethics committee):		
4.	Does your research involve participants who are unable to give informed consent, for example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	No
5.	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	No
6.	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	No
7.	Does your research involve participants disclosing information about sensitive subjects?	No
8.	Does your research involve the researcher travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (http://www.fco.gov.uk/en/)	No
9.	Does your research involve invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	No
10.	Does your research involve animals?	No
11.	Does your research involve the administration of drugs, placebos or other substances to study participants?	No
If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be		

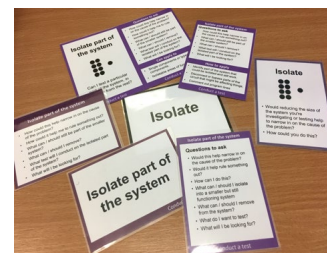
referred to the Senate Research Ethics Committee.		
12.	Does your research involve participants who are under the age of 18?	No
13.	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	No
14.	Does your research involve participants who are recruited because they are staff or students of City University London? For example, students studying on a particular course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	No
15.	Does your research involve intentional deception of participants?	No
16.	Does your research involve participants taking part without their informed consent?	No
17.	Does your research pose a risk to participants greater than that in normal working life?	No
18.	Does your research pose a risk to you, the researcher(s), greater than that in normal working life?	No
You must make a proportionate review application to the CSREC if your research involves human participation and you are not submitting any other ethics application (i.e. your answer to all questions 1 – 18 is “NO”).		

Part B: Ethics Proportionate Review Form

If you answered NO to all questions 1 – 18, you may use this part of the form to submit an application for a proportionate ethics review of your research. The form must be accompanied by all relevant information sheets, consent forms and interview/questionnaire schedules.

Note that all research participants should be fully informed about: the purpose of the research; the procedures affecting them or affecting any information collected about them, including information about what they will be asked to do, what data will be collected, how the data will be used, to whom it will be disclosed, and how long it will be kept; the fact that they can withdraw at any time without penalty.

Background Information	
Name:	Tracey Booth
Supervisor (if student):	Dr Simone Stumpf Dr Jon Bird
Your Research Project	
Title:	User review of design options for troubleshooting support materials aimed at novice Arduino users (CSREC180209TB)
Start date:	26/02/2018
End date:	01/10/2020
Describe your project: overall aim(s) and method (up to 300 words) Background My first study (S1) established that end-user developers (EUDs) experience numerous problems when developing physical computing prototypes with platforms like Arduino, and that circuit bugs are most likely to prevent successful development of a working prototype. In a subsequent, deeper analysis (S2) of data from the previous study, this time focusing on the natural troubleshooting behaviours of EUDs, I discovered that EUDs struggle to diagnose circuit bug-related problems and evaluate whether attempted fixes are successful. This is often the result of poor troubleshooting strategies/tactics. Extending upon this work, the rest of my PhD research aims to determine the effects of providing <i>support materials</i> on EUDs' troubleshooting of circuit bugs. To this end, inspired by creativity support card decks (e.g. Thinkpak), I am currently designing a set of physical cards that contain information about different strategies/tactics that EUDs can employ when troubleshooting. This study This new study (S3) will elicit user feedback on the early-stage design of these materials. A number of design variants (pictured below) have been created, based on the findings from studies 1 & 2, as well as a review of the literature (program debugging; electronic circuit troubleshooting; card-based tools). In this study, to provide valuable user input into the design of the support materials, six novice Arduino users will be asked to review and rank these variants, focusing on specific aspects of the designs.	



This research, involving representative users from the target population, will inform my outstanding design decisions, resulting in a single set of support materials that will be used in my next study – an observation of their effect on EUDs’ hands-on troubleshooting of Arduino circuit bugs (S4– date tbd but ethics approval has already been granted).

Participants

Participants will be 6 adults (18+ years), who are novice Arduino users. They will be recruited through maker and university networks, using flyers, social networks (Twitter and Facebook), mailing lists and personal contacts. Participants must be relatively new to using Arduino (or any other physical computing development platform) and not consider themselves to be experts in electronics or programming. Recruitment respondents will be sent a detailed study information sheet and all potential participants will be screened for eligibility, including their age and physical computing development experience. I will not be recruiting anyone under the age of 18, and vulnerable participants will be screened out.

Procedure, incl. data gathering & analysis

Each participant will attend a one-hour-long session, to be held in the City Interaction Lab, at City, University of London, at a pre-arranged time convenient for them. Each session will involve 2 participants – participants will be randomly assigned to these pairs.

At the start of the session each participant will complete the informed consent form. They will then fill in a brief background questionnaire, capturing their demographic details and their experience of developing physical computing prototypes.

Participants will then verbally answer two questions about their use of Arduino. Then, working in pairs, they will undertake a series of design review exercises, guided by the facilitator (me), using a semi-structured topic guide, focusing on different aspects of the designs. In each exercise, participants will be asked to discuss and provide feedback on the design variants in terms of a specific aspect, and rank them according to their preferences. Aspects are:

- Physical format (Card size; Card orientation; Handling)
- Textual information (Types of information; Amount of information; Location of information)
- Visual design characteristics (Imagery / iconography; Use of colour; Typography)

The session will be video-recorded for later analysis, using an external video camera, capturing participants’ verbal comments about the designs, and any non-verbal activity, e.g. manipulation of the designs (please note provisions made in the next section for anonymity regarding the use of these recordings). Any additional notes or diagrams made by participants will be digitised following the session.

A thematic analysis of the recordings will be completed after all sessions have taken place. The findings will be used to refine the design of the support materials.

Data security & privacy

Once gathered, identifying data will be anonymised - participants will be represented by randomly assigned ID numbers. Participant names will not be associated with the recordings or any other data, and will not appear in any reports or presentations, including where any video clips or screenshots are used in which faces are shown.

When signing the consent form, participants can opt in for allowing their faces to be shown in any video clips or stills used in presentations or publications. If a participant does not opt in for this, but still opts in to allow recordings to be used in presentations or publications, their face will be pixelated or blurred in any such presentations or publications, to anonymise them.

All data will be password protected, stored securely, and backed up. Only myself, my supervisory team (Dr Simone Stumpf, Dr Jon Bird and Dr Sara Jones), and my examiners, will have access to the data. If a participant decides to withdraw from the study at any point, I will destroy any data already gathered from them.

Attachments (these must be provided if applicable):

Participant information sheet(s)	Yes
Consent form(s)	Yes
Questionnaire(s) – background questionnaire	Yes
Topic guide(s) for interviews and focus groups	Yes
Permission from external organisations (e.g. for recruitment of participants)	Not applicable

Appendix K. Card design focus groups participant information sheet



City, University of London
Northampton Square
London EC1N 2BS
United Kingdom
+44 (0)20 7040 5060

Research Study Participant Information Sheet

Title of study: User review of design options for troubleshooting support materials aimed at novice Arduino users

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it's important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. If there is anything that is not clear or if you would like more information, please ask us.

"What is the purpose of the study?"

I'm a PhD research student in the Centre for Human-Computer Interaction Design (HCID) at City, University of London. The main aim of my PhD research is to find out how people can be supported in learning and using physical computing platforms such as Arduino. This particular study will help us to determine how best to present troubleshooting support information to non-expert Arduino users.

"Why have I been invited?"

I am looking for adults who have at least some – but not extensive – experience of using Arduino. Ideally you would be relatively new to using Arduino (and any other physical computing development platform) and would not consider yourself to be an expert in either electronics or programming. You must also be at least 18 years old and be able to undertake the activities described below.

"Do I have to take part? What happens if I sign up and then change my mind?"

Participation is voluntary. If you decide to take part, you will be asked to sign a consent form, but are still free to withdraw at any time – you will not be penalised or disadvantaged in any way and any data we have collected from you will be destroyed.

"What will happen if I take part?"

We will schedule a session with me (Tracey Booth), to take place in March 2018 in the Centre for HCI Design at City, University of London (280 St John St, London, EC1V 4BP), at a time convenient for you.

The session will take approximately 1 hour and will comprise:

1. Completing a short background questionnaire
2. Answering a few verbal questions about your use of Arduino.
3. Working in a pair, with another participant (who you may not know), providing your opinion on particular aspects of a number of paper-based designs we have created. Ranking the designs according to your preferences.

The session will be video-recorded, including audio, so that I can review/analyse the videos after the session.

Participants who complete the study will be given a £10 Amazon gift voucher, as a small token of thanks for taking part.

Page 1 of 2

"What are the possible benefits of taking part?"

The goal of end-user development research is to empower end users to develop and adapt systems and devices themselves. By taking part in this study you will contribute to our knowledge of how to design support for users of platforms such as Arduino, particularly novices.

"Will my taking part in the study be kept confidential?"

Yes – only myself, my PhD supervisors (Dr Simone Stumpf, Dr Jon Bird and Dr Sara Jones) and my PhD examiners, will have access to any data gathered. Data will be anonymised once gathered and will be stored securely. Your name will not be associated with the recordings or any other data and will not appear in any publications or presentations. You will have the option to allow sections of the recordings (e.g. video stills) to be used in publications or presentations, including whether or not your face can be shown, but this is entirely up to you.

"What will happen to the results of the research study?"

This study will be written up to form part of my PhD thesis and the results will inform my next research study towards this. I also hope to submit my findings for academic publication in international journals and conferences in the field of human-computer interaction.

"What will happen when the research study stops?"

Data will be stored for 10 years – the standard retention time for university studies – and then destroyed.

"What if there is a problem?"

If you have any problems, concerns or questions about this study, you should ask to speak to a member of the research team. If you remain unhappy and wish to complain formally, you can do this through the University complaints procedure. To complain about the study, you need to phone 020 7040 3040. You can then ask to speak to the Secretary to Senate Research Ethics Committee and inform them that the name of the project is: **User review of design options for troubleshooting support materials aimed at novice Arduino users**

You could also write to the Secretary at:

Anna Ramberg
Secretary to Senate Research Ethics Committee
Research Office, E214
City, University of London
Northampton Square
London, EC1V 0HB

Email: Anna.Ramberg.1@city.ac.uk

City, University of London, holds insurance policies which apply to this study. If you feel you have been harmed or injured by taking part in this study, you may be eligible to claim compensation. This does not affect your legal rights to seek compensation. If you are harmed due to someone's negligence, then you may have grounds for legal action.

"Who has reviewed the study?"

This study has been approved by City University London Computer Science Research Ethics Committee (CSREC).

Further information and contact details

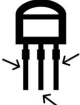
Researcher: Tracey Booth tracey.booth.1@city.ac.uk
PhD supervisor: Dr Simone Stumpf simone.stumpf.1@city.ac.uk +44 (0)20 7040 8168

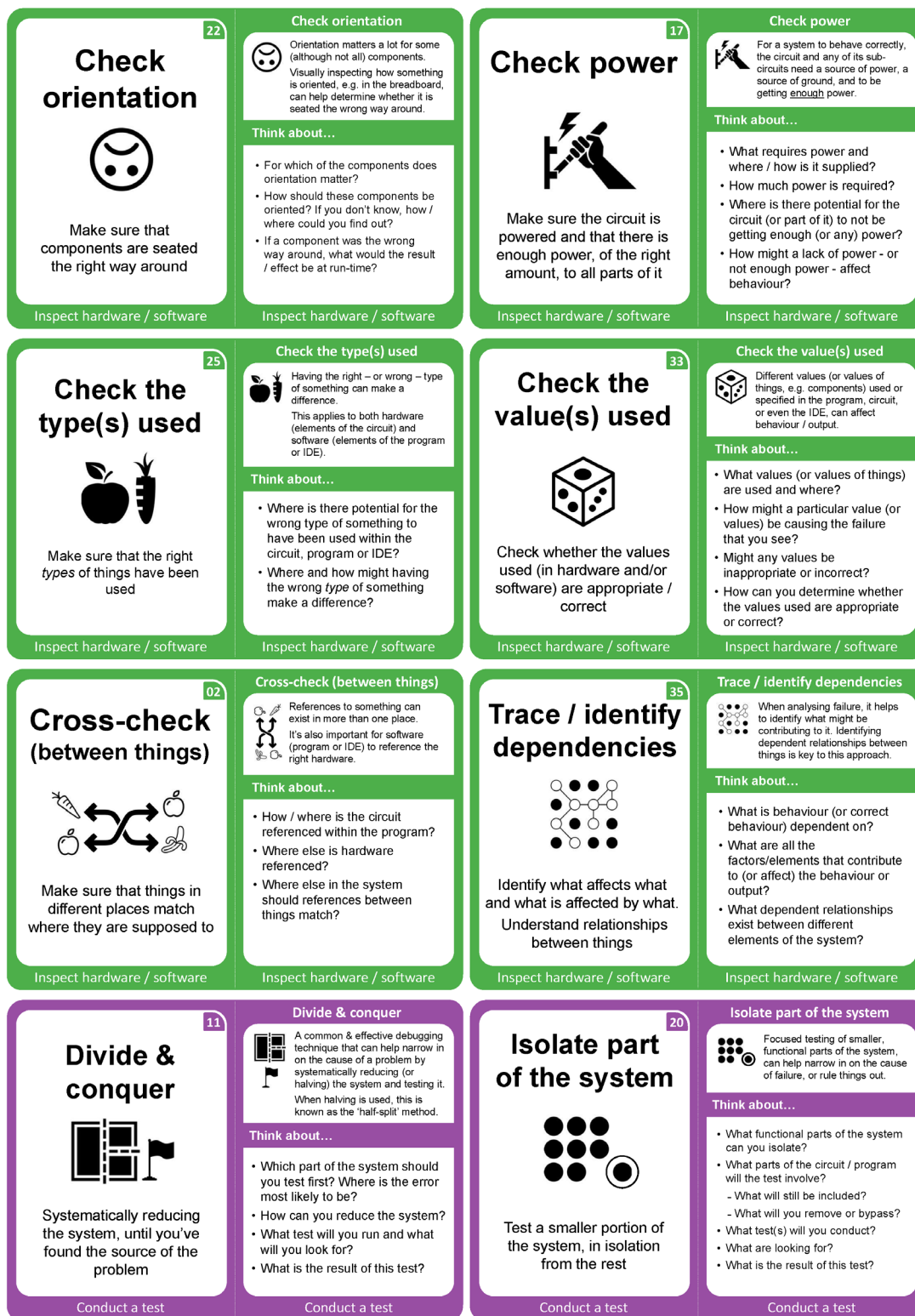
Page 2 of 2

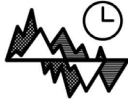
Appendix L. List of cards used in Study 2

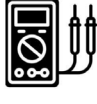
Tactic cards	
Analyse run-time behaviour/data	Analyse conditions Analyse frequency / consistency Analyse normality Analyse sequence Identify the symptoms Logging (print) statements Measure something Serial monitor
Conduct a test	Divide & conquer Isolate part of the system Redo (reimplement the same way) Reduce dependencies Restart Swap working & non-working Test for a faulty component
Get help	Compare to an example Read the requirements/specification View component / wiring information
Inspect hardware / software	Analyse the program/circuit Check circuit completeness Check component pinout Check for poor connections Check for special cases / uses Check if something's missing Check location of failure Check order (spatial) Check orientation Check power Check the type(s) used Check the value(s) used Cross-check (between things) Trace/identify dependencies
Stop... think	Consider alternatives Consider recent events Consider similar/familiar problems Question your assumptions
Best Practice cards	
Best practice	Avoid haphazard trial & error Diagnose, Fix, Evaluate result Keep track Make it easy to undo One 'fix' at a time Undo failed fixes

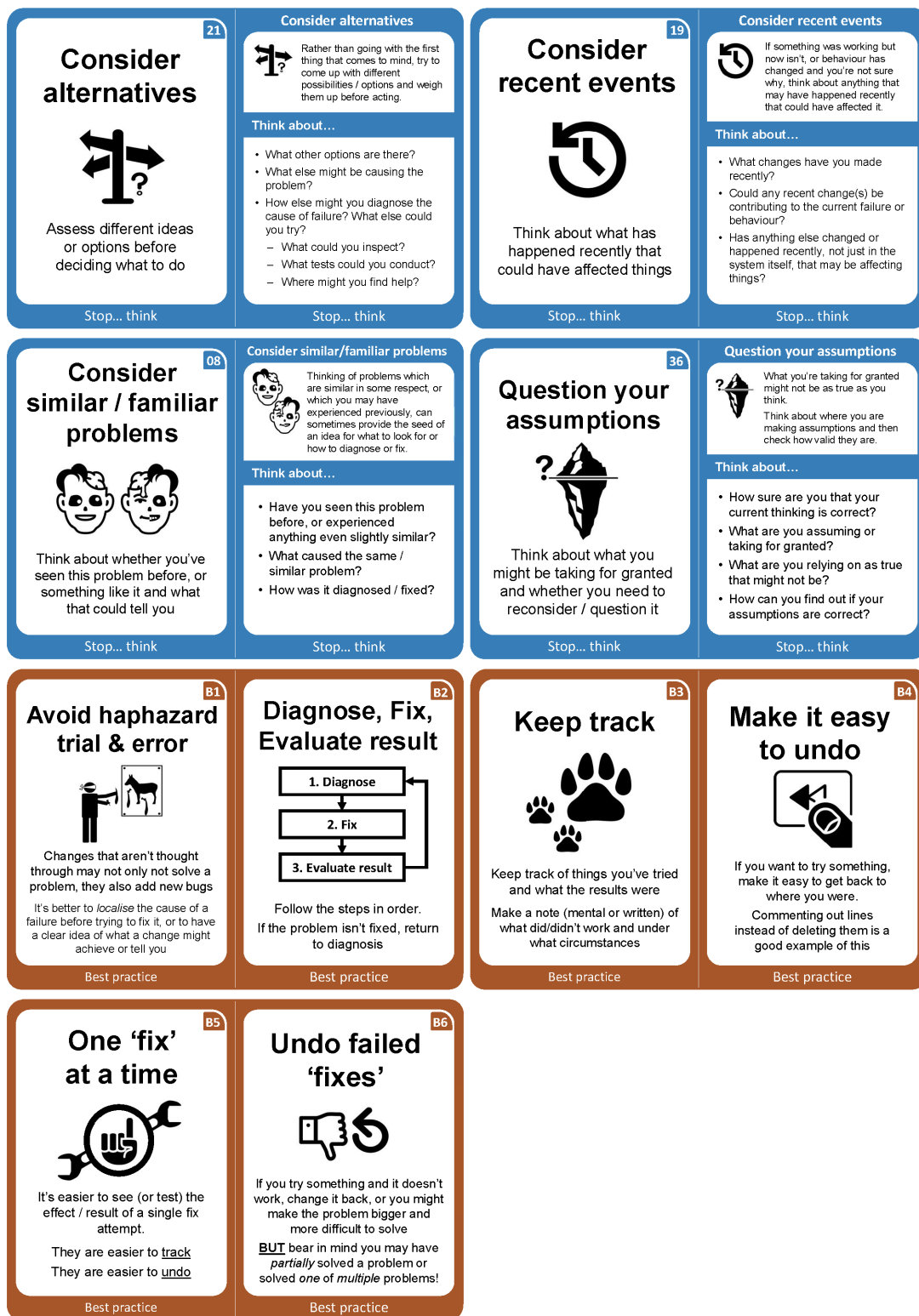
Appendix M. Tactics and Best Practice cards

Analyse the program / circuit ²⁴  Analyse the program, circuit or both, to see how it is constructed, what it does & how it works Inspect hardware / software	Analyse the program / circuit  Experienced developers often try to understand the system and how it functions, before deciding what troubleshooting action to take next. Think about... • Do you know what the system - or a particular part of it - consists of? Have you familiarised yourself with what is implemented? • Are you able to figure out how it works? The program? The circuit? How the program and circuit work together? • Does this provide you with any clues for what to do next? Inspect hardware / software	Check circuit completeness ⁰⁷  Make sure that the circuit (and all of its sub-circuits) are <u>complete</u> Inspect hardware / software	Check circuit completeness  For electrical current to flow within the circuit (or a particular part of it), there needs to be a continuous path between a source of power (e.g. power pin/rail or an output pin) and ground. Think about... • How can you determine whether the circuit - or the part that isn't working - is <i>complete</i>? – Where should you look? – What should you look at? – What should you look for? • What provides power within the circuit, and to what? • What sources of ground are there? Inspect hardware / software
Check component pinout ²⁷  Check whether the right types of connections have been used for a component Inspect hardware / software	Check component pinout  A pinout diagram shows the right types of connections for the contacts or pins (including those inside sockets) of an electrical component. Think about... • What connection types are required by the components in the circuit? • What effect might the wrong types of connections have on run-time functioning or behaviour? • Where / how might you find a pinout diagram for the components? Inspect hardware / software	Check for poor connections ¹⁰  Make sure all connections are properly seated and secure Inspect hardware / software	Check for poor connections  Poor connections (including mis-seated ones) can affect run-time behaviour. Visually inspecting connections can help identify any that aren't properly seated / secure. Think about... • How / where do components connect to the circuit? • What other types of connections do you have? • What connections might be loose or misaligned? • How else might a connection be 'poor'? Inspect hardware / software
Check for special cases / uses ³⁰  Check if there are any special cases or properties of things which might be having an effect Inspect hardware / software	Check special cases/uses  Some things have more than one function, or have special properties that which have a particular effect in specific circumstances. Think about... • Where in your circuit or program could a secondary or special use / property be causing an undesired effect? • How might you find out whether any secondary or special uses / properties exist for any part of the system? Inspect hardware / software	Check if something's missing ¹⁴  Check whether something important or necessary is missing from the system Inspect hardware / software	Check if something's missing  Failure may be caused by missing program elements, missing circuit components / connections, or even a missing cable or setting. Think about... • What should – or needs to – be in place, but isn't? • What program or circuit (or other) element might be missing? • What might be missing that could have led to the behaviour / data you're seeing? • If you're not sure, how can you find out? Inspect hardware / software
Check location of failure ⁰³  Look for errors in the part of the system where failure was observed Inspect hardware / software	Check location of failure  Sometimes it can help to see if there is anything obviously wrong with the system in the location where the failure is visible / occurs. Think about... • Where do you see evidence of failure? • What, in this location, might be causing the type of failure you have seen? • How can you establish whether something in this location is causing the failure? Inspect hardware / software	Check order (spatial) ²⁹  Make sure that things are connected or implemented in the right order Inspect hardware / software	Check order (spatial)  Order can matter. Visually inspecting the order (spatial sequence) in which things are connected or located, can help to identify whether something is in the wrong place. Think about... • In what order should things be connected or located? • Where does / doesn't order matter? • Might the order of something be affecting run-time behaviour or output? How? Inspect hardware / software



Redo (reimplement the same way)  Reimplement something in exactly the same way, to see if it fixes the problem Conduct a test	Redo (reimplement the same way) Errors aren't always easily visible to the eye. Re-implementing something can sometimes resolve a 'hidden' error. Think about... • What could you redo (reimplement)? • What hidden errors might there be and why/how might reimplementation resolve them? • Are there any risks to doing this? How might you avoid them? • What is the result? Conduct a test	Reduce dependencies  Reduce the number of factors potentially affecting / influencing behaviour or output Conduct a test	Reduce dependencies Reducing the number of factors that could be affecting behaviour or output can help isolate the cause of failure. Think about... • What is (or may be) affecting things? • What dependencies could you remove? • What is/isn't crucial for functioning? • How can you reduce (or bypass) dependencies? • What test(s) will you conduct? • What will you be looking for? • What is the result? Conduct a test
Restart  Try restarting (or reopening) something, to see if the problem disappears Conduct a test	Restart Restarting something can sometimes 'fix' a problem. "Turn it on and off again" is a well-known example of this tactic. Think about... • What could you restart and how? • What could you reopen? • Why do you think this might 'fix' your problem? • What is the result of this test? • If the problem still exists, what else might this tell you? Conduct a test	Swap working & non-working  Use something that works to test something that doesn't work Conduct a test	Swap working & non-working This can help you find out if a component is faulty, or if there's something wrong with a particular part of the circuit. Think about... • What type of swap will be useful? What makes most sense? - Swapping in a <u>working</u> component, or - Swapping in a <u>non-working</u> component? • What is the result of the test? Conduct a test
Test for a faulty component  Conduct a test to see if a component is faulty / broken Conduct a test	Test for a faulty component A faulty or broken component can affect behaviour / output. Testing can establish whether this is happening, or rule it out. Think about... • What might be faulty or broken? • What might have failed? • What might the effect of a faulty component be? How might you see / notice it? • What test(s) could you perform to establish - with certainty - whether a component is working? • What is the result of the test? Conduct a test	Analyse conditions  Look at the conditions (circumstances) under which behaviour or output occurs Analyse run-time behaviour / data	Analyse conditions To determine whether there is a problem, or diagnose a potential cause of one, it can be helpful to identify the conditions (circumstances) under which certain things occur. Think about... • What conditions (circumstances) can (or should) you check? • How can you create conditions in order to analyse the outcome? • Under what conditions <u>does</u> certain behaviour / output occur? • Under what conditions <u>doesn't</u> certain behaviour / output occur? Analyse run-time behaviour / data
Analyse frequency / consistency  Look at the pattern of behaviour or output over time Analyse run-time behaviour / data	Analyse frequency / consistency To determine whether there is (still) a problem, or diagnose a potential cause, it can be helpful to identify the pattern of behaviour / output over time. Think about... • How often is behaviour/output correct/incorrect? Always? Sometimes? Never? • Is the behaviour/output (or pattern), consistent? • If it's incorrect, is there a pattern over time? • Does the pattern tell you anything? Analyse run-time behaviour / data	Analyse normality  Determine whether (and how) behaviour / output is normal or abnormal Analyse run-time behaviour / data	Analyse normality It can be helpful to establish whether behaviour / output (or pattern thereof) is what you <u>should</u> be seeing, under the circumstances. Think about... • What should you expect to see? What would be normal (or correct)? • What would be abnormal (or incorrect)? • If you think something is abnormal or wrong, what evidence of this do you have? • If behaviour or output is abnormal, does it provide any clues as to why it is happening? Analyse run-time behaviour / data

12 Analyse sequence  Look at the sequence of behaviour or output Analyse run-time behaviour / data	Analyse sequence To determine whether there is a problem, or diagnose a potential cause of one, it can help to identify the sequence in which behaviour / output occurs. Think about... • What sequence do you expect? • Is this what you see? • If the sequence is incorrect / unexpected, is there a pattern to it? • Does the pattern tell you anything? Analyse run-time behaviour / data	01 Identify the symptoms  Recognise (and be able to describe) what failure looks like - what is and isn't working Analyse run-time behaviour / data	Identify the symptoms Before trying to fix a problem, try to determine exactly what the failure looks like. What is and isn't working? What are the symptoms that tell you there is a problem? Think about... • What <i>is</i> happening that <i>shouldn't</i> be happening? • What <i>isn't</i> happening that <i>should</i> be happening? • When / how / where <i>does</i> this occur? • When / how / where <i>doesn't</i> this occur? Analyse run-time behaviour / data
34 Logging (print) statements  Use 'print' statements to output hidden run-time data to the Serial Monitor Analyse run-time behaviour / data	Logging (print) statements You can use 'print' statements in your program to log (output) data, at run-time, to the Arduino IDE's Serial Monitor. You can then analyse this data. Think about... • What data would be useful to see that is otherwise hidden? • Where will you get this data? What will you output? • When viewing this data, what will you look for? • Is any additional information required for you to be able to interpret the data? Analyse run-time behaviour / data	04 Measure something  Use a tool to expose hidden data and/or see if something is working as expected Analyse run-time behaviour / data	Measure something You can use tools to measure part(s) or aspects of the circuit at run-time, making otherwise 'hidden' data visible. Think about... • What could you measure? • How can you measure it? • What will you look for? • What values would you expect to see? • If you see different values, what does this mean? • If you don't understand the values you see, how can you determine what they mean? Analyse run-time behaviour / data
37 Serial Monitor  Use the IDE's Serial Monitor window to view data outputted by 'print' statements in the program Analyse run-time behaviour / data	Serial Monitor The Serial Monitor - a separate window in the IDE - displays output (data) generated at run-time by 'Print' statements in the program. This otherwise hidden data can then be analysed. Think about... • Do statements in the program output data to the Serial Monitor? What data will they output? • When viewing output data in the Serial Monitor, what will you look for? • Could/does the output data tell you anything useful? • Is any additional information required for you to be able to interpret the data? Analyse run-time behaviour / data	26 Compare to an example  Use an example to judge whether something is correct Get help	Compare to an example Comparing what's been built to an example can help you to judge whether it has been implemented correctly. Think about... • What example would be useful? • What <i>type</i> or <i>format</i> of example would be most helpful? • How/where might you find it? • How will you judge whether an example you find is <i>appropriate</i>? • How will you know if an example you find is <i>correct</i>? Get help
13 Read the requirements / specification  Understand what things are supposed to be like - what should be in place and how it should behave Get help	Read the requirements / spec Understanding how things are supposed to be, can help you to determine what isn't correct. Think about... • How is it <i>supposed</i> to work? What is the <i>desired</i> or <i>expected</i> behaviour? • What is it supposed to look like? What should be there? • Does any of this differ from what you see? In what way? Get help	06 View component / wiring info  Find out the properties or characteristics of components or parts of the circuit, and how to use them Get help	View component/wiring info Supplement your own electronics knowledge with facts & guidance from other sources, for example, information about components or circuits in general. Think about... • What components or parts of the circuit do you need information about? • What type of information might be useful and why? • What properties / characteristics might be helpful to know? • Where might you find this information? Get help



Appendix N. Study 2 Ethics application

Ethics Proportionate Review Application: Staff and Research Students Computer Science Research Ethics Committee (CSREC)

Staff and research students in the Department of Computer Science undertaking research that involves human participation must apply for ethical review and approval before the research can commence. If the research is low-risk, an application can be submitted for a proportionate review using this form. Applicants are advised to read the information in the SMCSE Framework for Delegated Authority for Research Ethics prior to submitting an application.

There are two parts:

Part A: Ethics Checklist. The checklist determines whether the research is low-risk. If it is, Part B of the form should also be completed. If not, the checklist provides guidance as to where approval should be sought, but the checklist itself does not need to be submitted.

Part B: Ethics Proportionate Review Form. This part is the application for ethical approval of low-risk research and should only be completed if the answer to all questions (1 – 18) is NO.

Completed forms should be returned to the Chair of CSREC by email (*email address redacted*).

Part A: Ethics Checklist

If your answer to any of the following questions (1 – 3) is YES, you must apply to an appropriate external ethics committee for approval:		
1.	Does your research require approval from the National Research Ethics Service (NRES)? (E.g. because you are recruiting current NHS patients or staff? If you are unsure, please check at http://www.hra.nhs.uk/research-community/before-you-apply/determine-which-review-body-approvals-are-required/)	No
2.	Will you recruit any participants who fall under the auspices of the Mental Capacity Act? (Such research needs to be approved by an external ethics committee such as NRES or the Social Care Research Ethics Committee http://www.scie.org.uk/research/ethics-committee/)	No
3.	Will you recruit any participants who are currently under the auspices of the Criminal Justice System, for example, but not limited to, people on remand, prisoners and those on probation? (Such research needs to be authorised by the ethics approval system of the National Offender Management Service.)	No
If your answer to any of the following questions (4 – 11) is YES, you must apply to the Senate Research Ethics Committee for approval (unless you are applying to an external ethics committee):		
4.	Does your research involve participants who are unable to give informed consent, for example, but not limited to, people who may have a degree of learning disability or mental health problem, that means they are unable to make an informed decision on their own behalf?	No
5.	Is there a risk that your research might lead to disclosures from participants concerning their involvement in illegal activities?	No
6.	Is there a risk that obscene and or illegal material may need to be accessed for your research study (including online content and other material)?	No
7.	Does your research involve participants disclosing information about sensitive subjects?	No
8.	Does your research involve the researcher travelling to another country outside of the UK, where the Foreign & Commonwealth Office has issued a travel warning? (http://www.fco.gov.uk/en/)	No
9.	Does your research involve invasive or intrusive procedures? For example, these may include, but are not limited to, electrical stimulation, heat, cold or bruising.	No
10.	Does your research involve animals?	No
11.	Does your research involve the administration of drugs, placebos or other substances to study participants?	No
If your answer to any of the following questions (12 – 18) is YES, you must submit a full application to the Computer Science Research Ethics Committee (CSREC) for approval (unless you are applying to an external ethics committee or the Senate Research Ethics Committee). Your application may be referred to the Senate Research Ethics Committee.		

12.	Does your research involve participants who are under the age of 18?	No
13.	Does your research involve adults who are vulnerable because of their social, psychological or medical circumstances (vulnerable adults)? This includes adults with cognitive and / or learning disabilities, adults with physical disabilities and older people.	No
14.	Does your research involve participants who are recruited because they are staff or students of City University London? For example, students studying on a particular course or module. (If yes, approval is also required from the Head of Department or Programme Director.)	No
15.	Does your research involve intentional deception of participants?	No
16.	Does your research involve participants taking part without their informed consent?	No
17.	Does your research pose a risk to participants greater than that in normal working life?	No
18.	Does your research pose a risk to you, the researcher(s), greater than in normal working life?	No
You must make a proportionate review application to the CSREC if your research involves human participation and you are not submitting any other ethics application (i.e. your answer to all questions 1 – 18 is “NO”).		

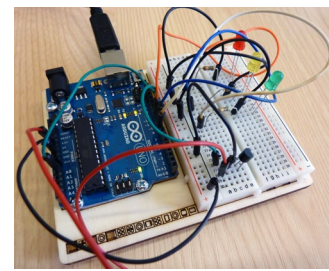
Part B: Ethics Proportionate Review Form

If you answered NO to all questions 1 – 18, you may use this part of the form to submit an application for a proportionate ethics review of your research. The form must be accompanied by all relevant information sheets, consent forms and interview/questionnaire schedules.

Note that all research participants should be fully informed about: the purpose of the research; the procedures affecting them or affecting any information collected about them, including information about what they will be asked to do, what data will be collected, how the data will be used, to whom it will be disclosed, and how long it will be kept; the fact that they can withdraw at any time without penalty.

Background Information	
Name:	Tracey Booth
Supervisor (if student):	Dr Simone Stumpf
Your Research Project	
Title:	Exploring how to support end-user developers in troubleshooting physical computing bugs
Start date:	01/07/2017
End date:	01/10/2020
<i>Describe your project: overall aim(s) and method (up to 300 words)</i> Physical computing development involves the construction and programming of microcontroller-based prototypes that interact with the world through sensors (e.g., light or temperature) and actuators (e.g. motors or LEDs). It therefore requires knowledge and skill in both electronics and programming, however, the Maker Movement has enticed many <i>end users lacking this expertise</i> into physical computing development. While platforms such as Arduino have been developed, ostensibly, to make physical computing development easier for end-user developers (EUDs), my first PhD study discovered that EUDs experience numerous problems when developing prototypes. A deeper analysis of the same data found that several participants had considerable difficulty <i>troubleshooting circuit bugs</i> – the main cause of task failure in that study. Therefore, this next study – a formative, empirical user study - aims to determine the effect of providing support materials, in the form of information about troubleshooting strategies/tactics and components, on end-user developers' troubleshooting (diagnosing, fixing and testing) of circuit bugs in physical computing prototypes, with a view to discovering what does and doesn't help. Participants Participants will be 20 adults (18+ years), of varying background and ability, who use the Arduino platform to develop physical prototypes for personal use. They will be recruited via hackerspaces and other Maker community groups through flyers and mailing lists, and through direct personal contacts. People who respond to the recruitment will be sent a detailed study information sheet. We will not be recruiting anyone under the age of 18, and vulnerable participants will be screened out. Procedure, incl. data gathering & analysis Each participant will individually attend an hour-and-a-half-long session at a pre-arranged time convenient for them. At the start of the session, they will complete the informed consent form. They will then fill in an online background questionnaire, capturing their demographic details as well as data about their experience in developing physical computing prototypes.	

Following this, in the main part of the session—the tasks—we will present a series of “buggy” physical computing prototypes to participants and ask them to find the bugs and fix them. The prototypes will be the same as that used in study 1, i.e. the “Love-o-meter” project in the official Arduino starter kit, which lights up 3 LEDs in response to body temperature when a temperature sensor is held. Fig1 shows an image of this prototype.



As participants work on these tasks, we will give them support materials and observe how these support materials affect participants’ troubleshooting, the outcomes of troubleshooting actions, and which types of information provided are used and at which points.

Participants will be asked to ‘think aloud’ (verbal protocol) when undertaking the tasks, and specifically to articulate any troubleshooting-related thoughts and decisions, including any use of the support materials provided. The session will be video-recorded for later analysis, using a combination of screen-recording software (to record all on-screen activity) and two external video cameras, each recording a different view: 1) close-up view of the prototype and 2) wider view of the desk and support materials, also capturing participants’ faces (note provisions made in the next section for anonymity regarding use of recordings).

After each task, participants will complete a very short questionnaire in which they rate the difficulty of the task and their confidence in successfully completing it. Participants will not be told how successful they were in finding or fixing the bugs. When all tasks have been undertaken, participants will complete a questionnaire in which they will rate the usefulness of the support materials provided, including the types of information presented. We will also hold a short debriefing interview to probe them for further detail on these responses and their use of the support materials. The interviews will be video-recorded and later analysed for comments.

Data security & privacy

Once gathered, identifying data will be anonymised - participants will be represented by randomly assigned ID numbers. Participant names will not be associated with the recordings or any other data, and will not appear in any reports or presentations, including where any video clips or screenshots are used in which faces are shown.

When signing the consent form, participants can opt in for allowing their faces to be shown in any video clips or stills used in presentations or publications. If a participant does not opt in for this, but still opts in to allow recordings to be used in presentations or publications, their face will be pixelated or blurred in any such presentations or publications, to anonymise them.

All data will be password protected, stored securely, and backed up. Only myself, my supervisory team (Dr Simone Stumpf, Dr Sara Jones, Dr Jon Bird), and my examiners, will have access to the data. If a participant decides to withdraw from the study at any point, I will destroy any data already gathered from them.

Attachments (these must be provided if applicable):

Participant information sheet(s)	Yes
Consent form(s)	Yes
Questionnaire(s) – background questionnaire, self-efficacy questionnaire, ‘usefulness’ questionnaire	Yes
Topic guide(s) for interviews and focus groups	Yes
Permission from external organisations (e.g. for recruitment of participants)	n/a

Research Study Amendments

Computer Science Research Ethics Committee (CSREC)

This form should be used to submit an amendment to research previously approved by the Computer Science Research Ethics Committee (CSREC), City University London. Completed forms should be returned to the Chair of CSREC by email.

Principal Investigator and Study Duration

Name:	Tracey Booth (supervised by Simone Stumpf)
Email:	(redacted)
Title of study:	Exploring how to support end-user developers in troubleshooting physical computing bugs
Start Date:	04 June 2018 (sessions to start)
End Date:	01 October 2020

Research Amendments

Type of Amendment/s (tick as appropriate)

Research procedure/protocol (including research instruments)	X
Participation group	X
Information Sheet/s	X
Consent form/s	X
Other recruitment documents	
Sponsorship/collaborations	
Principal investigator/supervisor	
Extension to approval	
Other	

Details of amendments (give details of each of the amendments requested, state where the changes have been made and attach all amended and new documentation)

- Participation group is now *novice* Arduino users, rather than just Arduino users. (**Information Sheet**)
- Participant sessions estimated to take 2 hours, rather than 1.5 hours. (**Information Sheet**)
- Study sessions will now take place in June 2018. (**Information Sheet**)
- Analysis has increased focus on reflection, requiring changes to questions in research instruments (**Support materials questionnaire; Debriefing interview topic guide**)
- **Consent form** updated to the format approved for my previous study (CSREC180209TB)

Justification for amendments

- As the population most likely to benefit from troubleshooting support materials is *novice* (i.e. non-expert) Arduino users, and the support materials have been designed specifically for them, recruitment will now specifically target this subgroup of the original population. The Information Sheet has been updated to reflect this.
- Two pilot runs of this study confirmed that participant sessions are more likely to take 2 hours. I believe it is better to adjust the stated time to the more accurate estimate than rush the sessions or remove/reduce parts of the procedure. The Information Sheet has been changed to reflect the revised estimate.
- The pilots also suggested that the study would benefit from some further design work on the support (intervention) materials. Recruitment of participants was therefore delayed until after this work had taken place. This included another study (CSREC180209TB conducted March 2018) in which design options for the support materials were reviewed by representatives of the target audience. The Information Sheet has been changed to reflect the new date of participant sessions.
- Qualitative analysis will now have an increased focus on *reflection*, requiring some changes to questions in the Support Materials Rating questionnaire and the post-task Debriefing Interview Topic Guide.
- For my previous study (Mar 2018) I changed the consent form in response to some feedback from CSREC. I have also applied these same changes to the consent form for *this* study.

If an extension is requested, specify the period

n/a

Other information (provide any other information which you believe should be taken into account during ethical review of the proposed changes)

Please note that although the participant group has changed, it is now merely a subgroup of the original group, rather than a different group entirely.

Declaration (to be signed by the Principal Investigator)

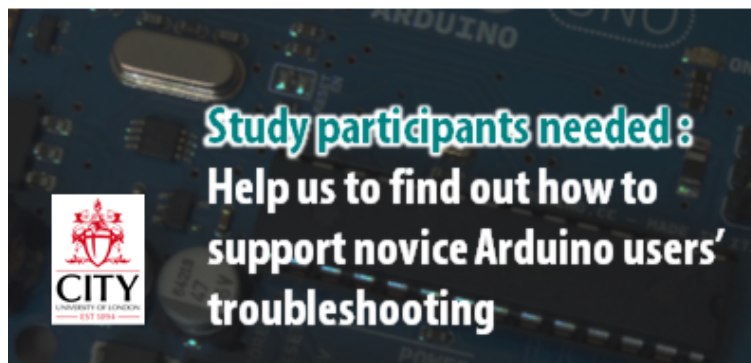
I certify that to the best of my knowledge the information given above, together with any accompanying information, is complete and correct and I take full responsibility for it.

	Signature	Signature
Principal Investigator(s) (student and supervisor if student project)	[Signature removed]	[Signature removed]
Date	15/05/18	

Appendix O. Study 2 Recruitment flyer



Relatively new to Arduino?



Study participants needed:
Help us to find out how to
support novice Arduino users'
troubleshooting



My PhD, in the Centre for Human-Computer Interaction Design at City, University of London, is exploring how to support novice Arduino users in overcoming development problems.

I am looking for adults (18+) who are relatively new to Arduino, to participate in a study in early June 2018



To find out more, please visit:
<http://tinyurl.com/ArduinoCity>



Appendix P. Study 2 Participant information sheet



City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom
+44 (0)20 7040 5060

Research Study Participant Information Sheet

Title of study:

Exploring how to support end-user developers in troubleshooting physical computing bugs

We would like to invite you to take part in a research study. Before you decide whether you would like to take part it's important that you understand why the research is being done and what it would involve for you. Please take time to read the following information carefully and discuss it with others if you wish. If there is anything that is not clear or if you would like more information, please ask us.

What is the purpose of the study?

I'm a PhD research student in the Centre for Human-Computer Interaction Design (HCID) at City, University of London. The main aim of my research is to find out how people can be supported in learning and using physical computing platforms such as Arduino. This particular study will help us to determine how to support non-expert Arduino users in overcoming problems they encounter during development. We have designed some troubleshooting support materials and want to see what happens when people use them.

Why have I been invited?

I am looking for adults who have some, but not extensive experience of using Arduino. As an eligible participant, you are not an expert in Arduino or any other physical computing development platform and would also not consider yourself to be an expert in either electronics or programming.

You must also be at least 18 years old and be able to undertake the activities described below.

Do I have to take part? What happens if I sign up and then change my mind?

Participation is voluntary. If you decide to take part, you will be asked to sign a consent form, but are still free to withdraw at any time - you will not be penalised or disadvantaged in any way and any data we have collected from you will be destroyed.

What will happen if I take part?

We will schedule a session with me (Tracey Booth), to take place in the Centre for HCI Design at City, University of London (280 St John St, London, EC1V 4BP), at a time convenient for you.

The session will take approximately 2 hours and will comprise:

1. A background questionnaire
2. Two short hands-on troubleshooting tasks using Arduino (Arduino UNO + the official Arduino development environment) with some provided support materials.
3. Rating the tasks and support materials
4. A short debriefing interview

The session will be video-recorded, including audio, so that I can review/analyse the videos after the session, but your participation will be kept confidential.

Participants who complete the study will be given a £20 Amazon gift voucher, as a small token of thanks for taking part.

What are the possible benefits of taking part?

The goal of end-user development research is to empower end users to develop and adapt systems and devices themselves. By taking part in this study you will contribute to the knowledge of how to design support for users of platforms such as Arduino, particularly novices / non-experts.

Will my taking part in the study be kept confidential?

Yes - only myself, my PhD supervisors (Dr Simone Stumpf, Dr Jon Bird and Dr Sara Jones) and my PhD examiners, will have access to any data gathered. Data will be anonymised once gathered and will be stored securely. Your name will not be associated with the recordings or any other data and will not appear in any publications or presentations. You will have the option to allow sections of the recordings (e.g. video stills) to be used in publications or presentations, but this is entirely up to you.

What will happen to the results of the research study?

This study will be written up to form part of my PhD thesis. I also hope to submit my findings for academic publication in international journals and conferences in the field of human-computer interaction.

What will happen when the research study stops?

Data will be stored for 10 years - the standard retention time for university studies - and then destroyed.

What if there is a problem?

If you have any problems, concerns or questions about this study, you should ask to speak to a member of the research team. If you remain unhappy and wish to complain formally, you can do this through the University complaints procedure. To complain about the study, you need to phone 020 7040 3040. You can then ask to speak to the Secretary to Senate Research Ethics Committee and inform them that the name of the project is: **Exploring how to support end-user developers in troubleshooting physical computing bugs**

You could also write to the Secretary at:

Anna Ramberg
Secretary to Senate Research Ethics Committee
Research Office, E214
City, University of London
Northampton Square
London, EC1V 0HB
Email: Anna.Ramberg.1@city.ac.uk

City, University of London, holds insurance policies which apply to this study. If you feel you have been harmed or injured by taking part in this study, you may be eligible to claim compensation. This does not affect your legal rights to seek compensation. If you are harmed due to someone's negligence, then you may have grounds for legal action.

Who has reviewed the study?

This study has been approved by City University London Computer Science Research Ethics Committee (CSREC).

Further information and contact details

Researcher: Tracey Booth tracey.booth.1@city.ac.uk
PHD supervisor: Dr Simone Stumpf simone.stumpf.1@city.ac.uk +44 (0)20 7040 8168

Thank you for taking the time to read this information sheet.

If you have any questions about the study, including about your eligibility to take part, please contact me at tracey.booth.1@city.ac.uk

Appendix Q. Study 2 Informed consent form



City, University of London
Northampton Square
London
EC1V 0HB
United Kingdom
+44 (0)20 7040 5060

INFORMED CONSENT

Study: Exploring how to support end-user developers in troubleshooting physical computing bugs

		Please initial
1.	I confirm that I have had the study explained to me, and I have read the participant information sheet, which I may keep for my records.	
	I understand that taking part in the study will involve: - Filling in a background questionnaire - Using provided equipment, including a computer, to troubleshoot bugs in physical computing prototypes - Using provided support materials when troubleshooting, as requested - Filling in ratings questionnaires - Being interviewed by the researcher - Allowing the session to be video-recorded, with audio.	
2.	I understand that data gathered will be held and processed for the following purpose(s): - PhD research - PhD assessment and examination - Research publications and presentations Public Task: The legal basis for processing your personal data will be that this research is a task in the public interest, that is City, University of London considers the lawful basis for processing personal data to fall under Article 6(1)(e) of GDPR (public task) as the processing of research participant data is necessary for learning and teaching purposes and all research with human participants by staff and students has to be scrutinised and approved by one of City's Research Ethics Committees.	
3.	I understand that any information I provide is confidential, and that no information that could lead to the identification of any individual will be disclosed in any reports on the project, or to any other party. No identifiable personal data will be published. The identifiable data will not be shared with any other organisation. I DO / DON'T consent to the use of sections (clips or stills) of the video recordings in publications and/or presentations (<i>please indicate applicable option</i>). I understand that my name would not accompany any such stills or clips. I DO / DON'T consent to my face being shown in sections (clips or stills) of any video recordings used in publications and/or presentations (<i>please indicate applicable option</i>). I understand that my name would not accompany any such stills or clips.	
4.	I understand that my participation is voluntary, that I can choose not to participate in part or all of the project, and that I can withdraw at any stage of the project without being penalised or disadvantaged in any way.	
5.	I agree to take part in this study.	

Name of Participant

Signature

Date

Name of Researcher

Signature

Date

Appendix R. Study 2 Background questionnaire

Background questionnaire

A) PERSONAL DETAILS

1. Participant ID

2. What is your age? In years, e.g. 37

3. What is your gender?

- ☐ Female
- ☐ Male
- ☐ _____

4. What is your current occupation / profession?

E.g. PhD student (Computer Science); University Lecturer (Maths); Interaction Designer

5. Have you ever been employed as a professional electronics engineer?

Yes No

6. Have you ever been employed as a professional programmer?

Yes No

B) EXPERTISE

7. How long, in total, have you been using Arduino?

Years Months

Time _____

8. How long, in total, have you been working with electronic circuits (in general)?

Years Months

Time _____

9. How long, in total, have you programming (in general)?

Years Months

Time _____

10. How expert do you think you are at using Arduino? (please circle applicable number)

Complete beginner 1 2 3 4 5 6 7 Complete expert

11. How expert do you think you are at working with electronic circuits (in general)?

Complete beginner 1 2 3 4 5 6 7 Complete expert

12. How expert do you think you are at programming (in general)?

Complete beginner 1 2 3 4 5 6 7 Complete expert

C) EXPERTISE IN TROUBLESHOOTING

13. How expert do you think you are at troubleshooting bugs in Arduino projects?

Complete beginner 1 2 3 4 5 6 7 Complete expert

14. How expert do you think you are at troubleshooting circuit bugs?

Complete beginner 1 2 3 4 5 6 7 Complete expert

15. How expert do you think you are at troubleshooting programming bugs?

Complete beginner 1 2 3 4 5 6 7 Complete expert

D) TRAINING

16. Have you had any training or instruction in Arduino (or a similar physical computing platform)

16.1 I have had training or instruction in Arduino (or similar)

Yes No

16.2 **What training / instruction?** *Please tick all that apply*

- ☐ Module(s) at university or other higher education institute
- ☐ High school class(es)
- ☐ Short workshop(s)
- ☐ Professional training courses (in-person)
- ☐ Online courses or structured tutorials
- ☐ Other - please give brief details _____

17. Have you had any training or instruction in electronic circuits?

17.1 I have had training or instruction in electronic circuits

Yes No

17.2 **What training / instruction?** *Please tick all that apply*

- ☐ Module(s) at university or other higher education institute
- ☐ High school class(es)
- ☐ Short workshop(s)
- ☐ Professional training courses (in-person)
- ☐ Online courses or structured tutorials
- ☐ Other - please give brief details _____

18. Have you had any training or instruction in programming?

18.1 I have had training or instruction in programming

Yes No

16.2 **What training / instruction?** *Please tick all that apply*

- ☐ Module(s) at university or other higher education institute
- ☐ High school class(es)
- ☐ Short workshop(s)
- ☐ Professional training courses (in-person)
- ☐ Online courses or structured tutorials
- ☐ Other - please give brief details _____

Appendix S. Study 2 Support Materials Questionnaire

1

Support Materials Evaluation Questionnaire

1. Participant ID

For each of the following questions, please circle one of the numbers (1-7)

Support material types

Generally, how useful were the following:

2. Troubleshooting tactics cards (Suggestions of tactics to try)

Not at all useful 1 2 3 4 5 6 7 Extremely useful

3. Categories (They group the tactics)

Not at all useful 1 2 3 4 5 6 7 Extremely useful

4. Play mat & rules (Mat upon which to place cards when troubleshooting, and constraints for use)

Not at all useful 1 2 3 4 5 6 7 Extremely useful

5. Card stand (Sectioned-stand that holds cards and displays categories)

Not at all useful 1 2 3 4 5 6 7 Extremely useful

Support materials format and general experience

6. How useful was it having the troubleshooting tactics in the form of playing cards?

Not at all useful 1 2 3 4 5 6 7 Extremely useful

7. What, specifically, did you like about the support materials?

8. Was there anything, specifically, that you didn't like about the support materials?

Effectiveness, Usability, Future usefulness

9. ... The support materials helped me find or fix at least one bug	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
10. ... The support materials gave me useful ideas for troubleshooting	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
11. ... The support materials helped me to remember things that I already knew	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
12. ... The support materials made me aware of new things that I didn't already know about	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
13. ... The support materials made troubleshooting more complicated	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
14. ... The support materials made me think/reflect more about what I was doing	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
15. ... The support materials helped to structure my troubleshooting	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
16. ... The support materials helped me to consider different hypotheses/ideas	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
17. ... The support materials were appropriate for the tasks	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
18. ... The support materials were easy to use	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
19. ... The support materials were confusing	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
20. ... The support materials were time-consuming to read	Strongly disagree	1	2	3	4	5	6	7	Strongly agree

21. ... I would have liked more detail/information in the support materials	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
22. ... I would have liked less detail/information in the support materials	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
23. ... The support materials were fun to use	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
24. ... The support materials make me feel more confident about Arduino troubleshooting	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
25. ... The support materials would be useful for novice Arduino users	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
26. ... The support materials would be useful for expert Arduino users	Strongly disagree	1	2	3	4	5	6	7	Strongly agree
27. ... I would like to use the support materials for future troubleshooting	Strongly disagree	1	2	3	4	5	6	7	Strongly agree

Appendix T. Study 2 Task instructions

Task Brief (without support)

Goal	
The Arduino project you've been given contains <u>bugs</u>. Your goal is to <u>find and fix bugs</u>. until the project <u>behaves as specified below</u> and contains <u>no bugs</u>.	
Behaviour specification (how it should work)	
The project uses a <u>sensor</u> to read temperature, and 3 LEDs to indicate levels of warmth as temperature changes. - No LEDs should be lit for the ambient room temperature - As temperature increases, <u>the LEDs should light up, one by one, until all 3 are lit</u> - As temperature decreases, the LEDs should <u>turn off, one by one, until the ambient room temperature is reached</u> and no LEDs are lit. You can <u>change the temperature by holding the sensor between your fingers</u>.	
Do ✓	Don't ✗
- Remember to 'think aloud' – say what's going through your mind – describe what you're thinking, what you're doing and any decisions you make. - You can use <u>built-in</u> help and examples or other online resources, <u>if/when you normally would</u>	Please don't search for the <u>exact</u> project, or a project that involves both a temperature sensor and <u>LEDs</u>, to copy it.
Time	
You have <u>up to 25 minutes</u> for this task.	

Task Brief (with support)

Goal	
The Arduino project you've been given contains <u>bugs</u>. Your goal is to <u>find and fix bugs</u>, using the support materials to help you. until the project <u>behaves as specified below</u> and contains <u>no bugs</u>.	
Behaviour specification (how it should work)	
The project uses a <u>sensor</u> to read temperature, and 3 LEDs to indicate levels of warmth as temperature changes. - No LEDs should be lit for the ambient room temperature - As temperature increases, <u>the LEDs should light up, one by one, until all 3 are lit</u> - As temperature decreases, the LEDs should <u>turn off, one by one, until the ambient room temperature is reached</u> and no LEDs are lit. You can <u>change the temperature by holding the sensor between your fingers</u>.	
Do ✓	Don't ✗
- You must use the support materials - Remember to 'think aloud' – say what's going through your mind while you're troubleshooting – describe what you're thinking, what you're doing and any decisions you make. - Describe what you're thinking and doing with the support materials, including which cards you're using/considering and what for - You can also use <u>built-in</u> help and examples or other online resources, <u>if/when you normally would</u>	Please don't search for the <u>exact</u> project, or a project that involves both a temperature sensor and <u>LEDs</u>, to copy it
Time	
You have <u>up to 25 minutes</u> for this task.	

Appendix U. Study 2 Interview topic guide

Debriefing interview (semi-structured) Topic guide

1.	What has their experience been with Arduino to date?	<i>V.brief warm up) e.g. Why started? How learnt? What done so far? How much/often used it? Had problems?</i>
2.	How did they find the tasks in this study? Easy, difficult...?	
3.	Did having the support materials make troubleshooting easier/quicker , or...?	<i>Perceived effectiveness; Usability. What and why?</i>
4.	Do they think it might have helped to have support materials for the other task ? Why?	<i>Did they prefer troubleshooting with/without support?</i>
5.	What did they think of the Support materials overall ?	<i>Tactics, best practice, etc. Which did they like / find useful (or not) etc</i>
6.	What did they think about having the Tactics & best practice presented as playing cards ?	<i>What did/didn't they like about the format?</i>
7.	What did they think of the Tactics in general ?	<i>As a whole. Were they easy to use? Easy to understand?</i>
8.	Were any of the tactics particularly useful ?	<i>Which and why?</i>
9.	What did they think about the way that information was presented, i.e. as questions rather than instructions .	<i>Was it helpful? Why do they think we took this approach?</i>
10.	What did they think about the playmat and rules for using the support materials?	<i>What did/didn't they like? Did they help/hinder? How?</i>
11.	Did they learn anything from the support materials? The tactics? Best practice? The troubleshooting steps?	<i>Probe for 'what' – specific tactics, knowledge; process; anything metacognitive)</i>
12.	Did the support materials have any effect on their thinking during the task?	<i>Probe for evidence of reflection; considering alternatives / different hypotheses; Did they prompt new ideas</i>
13.	Did the support materials have any effect on their behaviour during the task?	<i>Probe for whether it changed their troubleshooting process in any way</i>
14.	Was there anything that would/could have made the support materials more useful or easier to use for them specifically ?	
15.	Do they think they might like to have these materials when they are troubleshooting Arduino bugs in future ?	<i>Why [or why not]?</i>

Appendix V. Study 1A/1B Participant background data

Ptc	Age	Gender	Occupation	Physical computing				Programming				Electronics			
				Self-efficacy	Years	Expertise	Training	Years	Expertise	Training	Employed	Years	Expertise	Training	Employed
P01	27	Female	Post-Doctoral Researcher (HCI)	62	5.00	3	No	9.00	4	Yes	No	5.00	2	Yes	No
P02	27	Male	Broadcast Engineer	73	6.50	5	No	15.00	5	Yes	No	5.00	5	Yes	Yes
P03	22	Female	PhD Student (Computer Science)	52	1.00	6	No	8.00	7	Yes	Yes	1.00	5	No	No
P04	25	Female	PhD Student (Media & Arts)	67	2.25	4	Yes	4.00	3	Yes	No	0.00	2	No	No
P05	32	Female	Project Manager (Arts)	64	4.00	2	Yes	4.00	3	Yes	No	4.00	2	No	No
P06	46	Male	Events/Content producer	86	2.50	3	No	30.00	5	Yes	No	30.00	4	No	No
P07	30	Male	PhD student (Media & Arts Technology)	70	1.50	4	Yes	8.50	6	Yes	Yes	1.50	2	Yes	No
P08	33	Male	Restaurant owner	80	7.42	5	Yes	28.00	5	No	No	20.00	5	No	No
P09	29	Female	Director/Research Consultant (Tech & Arts)	82	4.58	5	Yes	4.58	4	Yes	No	4.58	5	Yes	No
P10	34	Female	Project Manager (Media & Technology)	68	2.50	3	Yes	2.50	2	Yes	No	6.00	2	Yes	No
P11	53	Male	High School Substitute Teacher (English Lit)	73	2.25	3	Yes	12.67	2	No	No	3.25	4	Yes	No
P12	41	Female	University Lecturer (Fashion Marketing)	41	2.00	2	Yes	15.00	5	Yes	Yes	1.00	1	No	No
P13	38	Female	Student (Science and Human Physiology)	68	0.00	2	No	10.00	2	Yes	No	15.00	2	Yes	No
P14	32	Male	Software Developer	73	5.00	4	Yes	11.00	6	Yes	Yes	5.00	3	No	No
P15	32	Male	Post-Doctoral Researcher (Computer Science)	70	0.50	3	No	16.00	6	Yes	Yes	16.00	5	No	No
P16	29	Male	Systems Analyst	84	1.67	4	Yes	7.00	6	Yes	Yes	1.67	2	No	No
P17	28	Male	PhD Student (HCI)	73	6.00	5	Yes	16.00	5	Yes	No	6.00	3	Yes	No
P18	30	Male	Education Programme Manager (Science)	74	4.00	3	No	5.00	4	No	No	4.00	2	No	No
P19	26	Male	Industrial & Web Designer	58	3.50	2	Yes	7.00	4	No	Yes	3.50	3	No	No
P20	22	Male	MSc Student (Comp.Sci & Embedded Systems)	76	2.50	4	Yes	4.50	4	Yes	No	2.50	3	Yes	No

Appendix W. Study 2 Participant background data

Ptc	Group	Age	Gender	Occupation	Arduino			Electronics			Employed	Programming			
					Years	Expertise	TS Expertise	Years	Expertise	TS expertise		Years	Expertise	TS Expertise	Employed
P110	NSWS	41	Male	Web Developer	0.92	2	2	0.92	1	1	No	8.00	6	4	Yes
P120	WSNS	28	Female	PhD Student (Media & Art tech)	0.50	4	4	1.50	3	3	No	3.00	5	5	No
P130	NSWS	26	Male	Masters Student (HCI)	0.92	3	2	0.92	2	2	No	2.42	6	6	Yes
P140	WSNS	39	Male	Electrician	0.25	4	4	1.00	4	4	No	1.00	2	2	No
P150	NSWS	21	Male	Undergrad. student (Biomedical Eng.)	0.25	3	3	2.17	4	5	No	0.42	3	3	No
P160	WSNS	21	Male	Undergrad. student (Engineering)	0.50	3	4	2.00	6	6	No	0.54	2	2	No
P170	NSWS	35	Female	Creative	1.00	3	3	1.50	3	3	No	2.17	4	3	No
P180	WSNS	51	Female	Sound engineer	0.33	2	2	0.33	1	1	No	0.17	2	2	No
P190	NSWS	21	Male	Undergrad. student (Computer Science)	0.92	2	3	0.67	5	1	No	3.17	6	6	Yes
P200	WSNS	38	Male	Charity consultant	0.17	2	2	0.17	2	2	No	5.00	5	4	No
P210	NSWS	31	Male	Lab technician	0.17	4	4	0.58	4	5	No	0.25	3	2	No
P220	WSNS	48	Female	Masters student (Computational Art)	0.42	3	2	0.42	2	2	No	0.83	2	2	No
P230	NSWS	20	Female	Undergrad. student (Mech. Eng)	0.08	2	3	2.00	4	3	No	0.50	3	2	No
P240	WSNS	51	Female	Masters student (Computational Art)	1.25	3	2	1.00	2	1	No	2.00	2	2	No
P250	NSWS	20	Female	Undergrad. student (Creative Computing)	0.17	2	1	0.42	2	1	No	1.00	3	3	No
P260	WSNS	28	Male	PhD Student (Media & Art Technology)	2.00	2	3	1.50	3	2	No	2.00	4	5	No
P270	NSWS	47	Female	Finance	6.00	2	2	1.00	2	2	No	4.00	3	3	No
P280	WSNS	21	Female	Masters student (Design)	2.00	3	2	0.17	4	1	No	0.42	4	1	No
P290	NSWS	37	Male	Freelance Educator (Primary school)	3.00	2	3	0.50	2	3	No	4.00	4	5	No
P300	WSNS	31	Female	Research Fellow (HCI)	6.00	3	2	2.00	3	2	No	10.00	5	5	Yes

TS: Troubleshooting

Bibliography

'123D Circuits Electronics Lab'. n.d. Autodesk 123D Circuits. Accessed 12 July 2015. <https://123d.circuits.io/lab>.

'About Max'. n.d. Cycling '74. Accessed 30 June 2016. <https://cycling74.com/products/max/>.

'Adafruit Customer Support Forums'. n.d. Adafruit. Accessed 29 June 2016. <https://forums.adafruit.com/>.

Agans, David J. 2002. *Debugging: The 9 Indispensable Rules for Finding Even the Most Elusive Software and Hardware Problems*. New York, NY, USA: American Management Assoc., Inc.

Aghaee, S., A. F. Blackwell, D. Stillwell, and M. Kosinski. 2015. 'Personality and Intrinsic Motivational Factors in End-User Programming'. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 29–36. <https://doi.org/10.1109/VLHCC.2015.7357195>.

Alchemer LLC. n.d. *Alchemer (Formerly SurveyGizmo)*. <https://www.alchemer.com/>.

Ananthanarayan, Swamy, Nathan Lapinski, Katie Siek, and Michael Eisenberg. 2014. 'Towards the Crafting of Personal Health Technologies'. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, 587–96. DIS '14. New York, NY, USA: ACM. <https://doi.org/10.1145/2598510.2598581>.

Anderson, Fraser, Tovi Grossman, and George Fitzmaurice. 2017. 'Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry'. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 331–42. Québec City QC Canada: ACM. <https://doi.org/10.1145/3126594.3126637>.

Antle, Alissa N., and Alyssa F. Wise. 2013. 'Getting down to Details: Using Theories of Cognition and Learning to Inform Tangible User Interface Design'. *Interacting with Computers* 25 (1): 1–20. <https://doi.org/10.1093/iwc/iws007>.

Arcila, Dave. 2013. 'Testing Every Aspect of Your Game Design with a Deck of Lenses'. *Envato Tuts+ | Game Development* (blog). 6 February 2013. <https://gamedevelopment.tutsplus.com/articles/testing-every-aspect-of-your-game-design-with-a-deck-of-lenses--gamedev-4232>.

'Arduino'. n.d. Accessed 17 January 2013. <http://www.arduino.cc/>.

'Arduino Forum'. n.d. Accessed 29 June 2016. <https://forum.arduino.cc/>.

'Arduino Starter Kit'. n.d. Arduino. Accessed 21 July 2015. <https://www.arduino.cc/en/Main/ArduinoStarterKit>.

Bandura, Albert. 1978. 'Reflections on Self-Efficacy'. *Advances in Behaviour Research and Therapy* 1 (4): 237–69. [https://doi.org/10.1016/0146-6402\(78\)90012-7](https://doi.org/10.1016/0146-6402(78)90012-7).

Banzi, Massimo. 2009. *Getting Started with Arduino*. 1 edition. Sebastopol, CA, USA: Make: Books, O'Reilly Media, Inc.

Barragán, Hernando. 2004. 'Wiring: Prototyping Physical Interaction Design'. Ivrea: Interaction Design Institute Ivrea. http://wiki.wiring.co/images/7/76/Wiring_thesis.pdf.

Barragán, Hernando. 2016. 'The Untold History of Arduino'. 2016. <https://arduinhistory.github.io/>.

Bates, Marcia J. 1990. 'Where Should the Person Stop and the Information Search Interface Start?' *Information Processing & Management* 26 (5): 575–91. [https://doi.org/10.1016/0306-4573\(90\)90103-9](https://doi.org/10.1016/0306-4573(90)90103-9).

- Beckwith, Laura A. 2007. 'Gender HCI Issues in End-User Programming'. Oregon State University. <http://ir.library.oregonstate.edu/xmlui/bitstream/handle/1957/4954/FinalVersion.pdf>.
- Bekker, Tilde, and Alissa N. Antle. 2011. 'Developmentally Situated Design (DSD): Making Theoretical Knowledge Accessible to Designers of Children's Technology'. In *Proceedings of the 2011 SIGCHI Conference on Human Factors in Computing Systems*, 2531–40. New York, NY, USA: ACM. <https://doi.org/10.1145/1978942.1979312>.
- Benchoff, Brian. 2015. 'Before Arduino There Was BASIC Stamp: A Classic Teardown'. *Hackaday* (blog). 27 August 2015. <https://hackaday.com/2015/08/27/before-arduino-there-was-basic-stamp-a-classic-teardown/>.
- Bergin, Susan, and Ronan Reilly. 2005. 'The Influence of Motivation and Comfort-Level on Learning to Program'. In *Proceedings of the 17th Workshop of the Psychology of Programming Interest Group*. University of Sussex, Brighton, UK.
- Bird, Jon, Paul Marshall, and Yvonne Rogers. 2009. 'Low-Fi Skin Vision: A Case Study in Rapid Prototyping a Sensory Substitution System'. In *Proceedings of the 23rd British HCI Group Annual Conference on People and Computers: Celebrating People and Technology*, 55–64. BCS-HCI '09. Swinton, UK, UK: British Computer Society. <http://0-dl.acm.org.wam.city.ac.uk/citation.cfm?id=1671011.1671018>.
- Blackwell, A.F. 2002. 'First Steps in Programming: A Rationale for Attention Investment Models'. In *IEEE 2002 Symposium on Human Centric Computing Languages and Environments, 2002. Proceedings*, 2–10. <https://doi.org/10.1109/HCC.2002.1046334>.
- Blikstein, Paulo. 2015. 'Computationally Enhanced Toolkits for Children: Historical Review and a Framework for Future Design'. *Foundations and Trends® in Human-Computer Interaction* 9 (1): 1–68. <https://doi.org/10.1561/11000000057>.
- Booth, Tracey, Jon Bird, Simone Stumpf, and Sara Jones. 2019. 'Designing Troubleshooting Support Cards for Novice End-User Developers of Physical Computing Prototypes'. In *End-User Development*, edited by Alessio Malizia, Stefano Valtolina, Anders Mørch, Alan Serrano, and Andrew Stratton, 191–99. Cham: Springer International Publishing.
- Booth, Tracey, and Simone Stumpf. 2013. 'End-User Experiences of Visual and Textual Programming Environments for Arduino'. In *End-User Development: Proceedings of the Fourth International Symposium on End-User Development (IS-EUD 2013)*, edited by Yvonne Dittrich, Margaret Burnett, Anders Mørch, and David Redmiles, 7897:25–39. Lecture Notes in Computer Science. Springer Berlin Heidelberg. http://dx.doi.org/10.1007/978-3-642-38706-7_4.
- Booth, Tracey, Simone Stumpf, Jon Bird, and Sara Jones. 2016. 'Crossed Wires: Investigating the Problems of End-User Developers in a Physical Computing Task'. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3485–97. CHI '16. New York, NY, USA: ACM. <https://doi.org/10.1145/2858036.2858533>.
- Boren, Ted, and Judith Ramey. 2000. 'Thinking Aloud: Reconciling Theory and Practice'. *Professional Communication, IEEE Transactions On* 43 (October): 261–78. <https://doi.org/10.1109/47.867942>.
- Boulay, Benedict du. 1986. 'Some Difficulties of Learning to Program'. *Journal of Educational Computing Research* 2 (1): 57–73. <https://doi.org/10.2190/3LFX-9RRF-67T8-UVK9>.
- Boulay, Benedict du. 1989. 'Some Difficulties of Learning to Program'. In *Studying the Novice Programmer*, edited by Elliot Soloway and James C. Spohrer, 283–300. Hillsdale, New Jersey, USA: Lawrence Erlbaum Associates, Inc.
- Brandt, Joel, Philip J. Guo, Joel Lewenstein, Mira Dontcheva, and Scott R. Klemmer. 2009. 'Two Studies of Opportunistic Programming: Interleaving Web Foraging, Learning, and Writing Code'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1589–98. CHI '09. New York, NY, USA: ACM. <https://doi.org/10.1145/1518701.1518944>.

Braun, Virginia, and Victoria Clarke. 2006. 'Using Thematic Analysis in Psychology'. *Qualitative Research in Psychology* 3 (2): 77–101. <https://doi.org/10.1191/1478088706qp063oa>.

Brueschke, Erich E, and Michael Mack. 2019. 'The History of the Heath Companies and Heathkits: 1909 to 2019'. *The AWA Review* 32: 125–64.

Buechley, Leah. 2005. *LilyPad Arduino: How an Open Source Hardware Kit Is Sparking New. Engineering and Design Communities*. MIT Media Lab, Cambridge. https://llk.media.mit.edu/courses/readings/democratized_LilyPad.pdf.

Buechley, Leah, Mike Eisenberg, and Nwanua Elumeze. 2007. 'Towards a Curriculum for Electronic Textiles in the High School Classroom'. In *Proceedings of the 12th Annual Conference on Innovation and Technology in Computer Science Education*, 39:28–32. Dundee, Scotland, United Kingdom. <https://doi.org/10.1145/1269900.1268795>.

Buechley, Leah, and Benjamin Mako Hill. 2010. 'LilyPad in the Wild: How Hardware's Long Tail Is Supporting New Engineering and Design Communities'. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, 199–207. DIS '10. New York, NY, USA: ACM. <https://doi.org/10.1145/1858171.1858206>.

Buechley, Leah, and Hannah Perner-Wilson. 2012. 'Crafting Technology: Reimagining the Processes, Materials, and Cultures of Electronics'. *ACM Trans. Comput.-Hum. Interact.* 19 (3): 21:1-21:21. <https://doi.org/10.1145/2362364.2362369>.

Burnett, Margaret M. 2009. 'What Is End-User Software Engineering and Why Does It Matter?' In *End-User Development*, edited by Volkmar Pipek, Mary Beth Rosson, Boris de Ruyter, and Volker Wulf, 15–28. Lecture Notes in Computer Science 5435. Springer Berlin Heidelberg. http://0-link.springer.com.wam.city.ac.uk/chapter/10.1007/978-3-642-00427-8_2.

Burnett, Margaret M., John Atwood, Rebecca Walpole Djang, James Reichwein, Herkimer Gottfried, and Sherry Yang. 2001. 'Forms/3: A First-Order Visual Language to Explore the Boundaries of the Spreadsheet Paradigm'. *Journal of Functional Programming* 11 (02): 155–206.

Burnett, Margaret M., Curtis Cook, and Gregg Rothermel. 2004. 'End-User Software Engineering'. *Communications of the ACM* 47 (9): 53–58. <https://doi.org/10.1145/1015864.1015889>.

Burnett, Margaret M., and Brad A. Myers. 2014. 'Future of End-User Software Engineering: Beyond the Silos'. In *Proceedings of the on Future of Software Engineering*, 201–11. FOSE 2014. New York, NY, USA: ACM. <https://doi.org/10.1145/2593882.2593896>.

Buur, Jacob, and Astrid Soendergaard. 2000. 'Video Card Game: An Augmented Environment for User Centred Design Discussions'. In , 63–69. ACM. <https://doi.org/10.1145/354666.354673>.

Cao, Jill. 2013. 'Helping End-User Programmers Help Themselves – the Idea Garden Approach'. Corvallis, OR, USA: Oregon State University.

Cao, Jill, Scott D. Fleming, and Margaret M. Burnett. 2011. 'An Exploration of Design Opportunities for "Gardening"; End-User Programmers' Ideas'. In *2011 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 35–42. <https://doi.org/10.1109/VLHCC.2011.6070375>.

Cao, Jill, Scott D. Fleming, Margaret Burnett, and Christopher Scaffidi. 2015. 'Idea Garden: Situated Support for Problem Solving by End-User Programmers'. *Interacting with Computers* 27 (6): 640–60. <https://doi.org/10.1093/iwc/iwu022>.

Cao, Jill, Irwin Kwan, Faezeh Bahmani, Margaret Burnett, Scott D. Fleming, Josh Jordahl, Amber Horvath, and Sherry Yang. 2013. 'End-User Programmers in Trouble: Can the Idea Garden Help Them to Help Themselves?' In *2013 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 151–58. <https://doi.org/10.1109/VLHCC.2013.6645260>.

Cao, Jill, Yann Riche, Susan Wiedenbeck, Margaret M. Burnett, and Valentina Grigoreanu. 2010. 'End-User Mashup Programming: Through the Design Lens'. In *Proceedings of the 28th International Conference on Human Factors in Computing Systems*, 1009–18. CHI '10. New York, NY, USA: ACM. <https://doi.org/10.1145/1753326.1753477>.

Carroll, John M. 1998. *Minimalism Beyond the Nurnberg Funnel*. Cambridge, MA, USA: MIT Press. <http://mitpress.mit.edu/books/minimalism-beyond-nurnberg-funnel>.

Carroll, John M., and Mary Beth Rosson. 1987. 'Paradox of the Active User'. In *Interfacing Thought: Cognitive Aspects of Human-Computer Interaction*, edited by John M. Carroll, 80–111. Cambridge, MA, USA: MIT Press. <http://dl.acm.org.wam.city.ac.uk/citation.cfm?id=28446.28451>.

Chang, Kerry Shih-Ping, and Brad A. Myers. 2016. 'Using and Exploring Hierarchical Data in Spreadsheets'. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 2497–2507. CHI '16. New York, NY, USA: ACM. <https://doi.org/10.1145/2858036.2858430>.

Compeau, Deborah R., and Christopher A. Higgins. 1995. 'Computer Self-Efficacy: Development of a Measure and Initial Test'. *MIS Quarterly* 19 (2): 189–211. <https://doi.org/10.2307/249688>.

Craft, Brock. 2013. 'Ten Troubleshooting Tips'. In *Arduino Projects For Dummies*, 1 edition, 359–67. Chichester, West Sussex, UK: John Wiley & Sons, Ltd.

Cressey, Daniel. 2017. 'The DIY Electronics Transforming Research'. *Nature News* 544 (7648): 125. <https://doi.org/10.1038/544125a>.

Creswell, John W., and J. David Creswell. 2018. *Research Design: Qualitative, Quantitative, and Mixed Methods Approaches*. 5th edition. Los Angeles: Sage Publications, Inc.

Creswell, John W., and Vicki L. Plano Clark. 2011. *Designing and Conducting Mixed Methods Research*. 2nd edition. Los Angeles: Sage Publications, Inc.

Cypher, Allen, Mira Dontcheva, Tessa Lau, and Jeffrey Nichols. 2010. *No Code Required: Giving Users Tools to Transform the Web*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

De Roeck, Dries, Karin Slegers, Johan Criel, Marc Godon, Laurence Claeys, Katriina Kilpi, and An Jacobs. 2012. 'I Would DiYSE for It!: A Manifesto for Do-It-Yourself Internet-of-Things Creation'. In *Proceedings of the 7th Nordic Conference on Human-Computer Interaction: Making Sense Through Design*, 170–79. NordiCHI '12. New York, NY, USA: ACM. <https://doi.org/10.1145/2399016.2399044>.

Deng, Ying, Alissa N. Antle, and Carman Neustaedter. 2014. 'Tango Cards: A Card-Based Design Tool for Informing the Design of Tangible Learning Games'. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, 695–704. New York, NY, USA: ACM. <https://doi.org/10.1145/2598510.2598601>.

DesPortes, Kayla, and Betsy DiSalvo. 2019. 'Trials and Tribulations of Novices Working with the Arduino'. In *Proceedings of the 2019 ACM Conference on International Computing Education Research*, 219–27. ICER '19. New York, NY, USA: ACM. <https://doi.org/10.1145/3291279.3339427>.

- Dougherty, Dale, Tim O'Reilly, and Ariane Conrad. 2016. *Free to Make: How the Maker Movement Is Changing Our Schools, Our Jobs, and Our Minds*. North Atlantic Books.
- Drew, Daniel, Julie L. Newcomb, William McGrath, Filip Maksimovic, David Mellis, and Björn Hartmann. 2016. 'The Toastboard: Ubiquitous Instrumentation and Automated Checking of Breadboarded Circuits'. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 677–86. UIST '16. Tokyo, Japan: ACM. <https://doi.org/10.1145/2984511.2984566>.
- Dumas, Joseph S., and Janice C. Redish. 1999. *A Practical Guide to Usability Testing*. Rev. ed.. Exeter: Intellect.
- Engelhardt, Paula Vetter, and Robert J. Beichner. 2004. 'Students' Understanding of Direct Current Resistive Electrical Circuits'. *American Journal of Physics* 72 (1): 98–115. <https://doi.org/10.1119/1.1614813>.
- Fields, Deborah A., Kristin A. Searle, and Yasmin B. Kafai. 2016. 'Deconstruction Kits for Learning: Students' Collaborative Debugging of Electronic Textile Designs'. In *Proceedings of the 6th Annual Conference on Creativity and Fabrication in Education*, 82–85. FabLearn '16. New York, NY, USA: ACM. <https://doi.org/10.1145/3003397.3003410>.
- Fitzgerald, Sue, Gary Lewandowski, Renée McCauley, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. 'Debugging: Finding, Fixing and Flailing, a Multi-Institutional Study of Novice Debuggers'. *Computer Science Education* 18 (2): 93–116. <https://doi.org/10.1080/08993400802114508>.
- Fleck, Rowanne, and Geraldine Fitzpatrick. 2010. 'Reflecting on Reflection: Framing a Design Landscape'. In *Proceedings of the 22nd Conference of the Computer-Human Interaction Special Interest Group of Australia on Computer-Human Interaction*, 216–23. New York, NY, USA: ACM. <https://doi.org/10.1145/1952222.1952269>.
- Fourney, Adam, and Michael Terry. 2012. 'PICL: Portable in-Circuit Learner'. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology - UIST '12*, 569. Cambridge, Massachusetts, USA: ACM Press. <https://doi.org/10.1145/2380116.2380188>.
- Frey, Bruce B. 2018. *The SAGE Encyclopedia of Educational Research, Measurement, and Evaluation*. 2455 Teller Road, Thousand Oaks, California 91320: SAGE Publications, Inc. <https://doi.org/10.4135/9781506326139>.
- Friedman, Batya, and David Hendry. 2012. 'The Envisioning Cards: A Toolkit for Catalyzing Humanistic and Technical Imaginations'. In *Proceedings of the 2012 SIGCHI Conference on Human Factors in Computing Systems*, 1145–48. New York, NY, USA: ACM. <https://doi.org/10.1145/2207676.2208562>.
- Gallacher, Sarah, Jenny O'Connor, Jon Bird, Yvonne Rogers, Licia Capra, Daniel Harrison, and Paul Marshall. 2015. 'Mood Squeezer: Lightening Up the Workplace Through Playful and Lightweight Interactions'. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, 891–902. CSCW '15. New York, NY, USA: ACM. <https://doi.org/10.1145/2675133.2675170>.
- Garner, Sandy, Patricia Haden, and Anthony Robins. 2005. 'My Program Is Correct but It Doesn't Run: A Preliminary Investigation of Novice Programmers' Problems'. In *Proceedings of the 7th Australasian Conference on Computing Education - Volume 42*, 173–80. ACE '05. Darlinghurst, Australia, Australia: Australian Computer Society, Inc. <http://0-dl.acm.org.wam.city.ac.uk/citation.cfm?id=1082424.1082446>.
- Gibb, Alicia M. 2010. 'New Media Art, Design, and the Arduino Microcontroller: A Malleable Tool'. Pratt Institute. <http://aliciagibb.com/wp-content/uploads/2013/01/New-Media-Art-Design-and-the-Arduino-Microcontroller-2.pdf>.
- Gick, Mary L. 1986. 'Problem-Solving Strategies'. *Educational Psychologist* 21 (1/2): 99.

- Good, Judith, and Kate Howland. 2017. 'Programming Language, Natural Language? Supporting the Diverse Computational Activities of Novice Programmers'. *Journal of Visual Languages & Computing*, Special Issue on Programming and Modelling Tools, 39 (April): 78–92. <https://doi.org/10.1016/j.jvlc.2016.10.008>.
- Greenberg, Saul, and Chester Fitchett. 2001. 'Phidgets: Easy Development of Physical Interfaces through Physical Widgets'. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, 209–18. UIST '01. New York, NY, USA: ACM. <https://doi.org/10.1145/502348.502388>.
- Grigoreanu, Valentina I., Margaret M. Burnett, and George G. Robertson. 2010. 'A Strategy-Centric Approach to the Design of End-User Debugging Tools'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 713–22. CHI '10. New York, NY, USA: ACM. <https://doi.org/10.1145/1753326.1753431>.
- Grigoreanu, Valentina I., Margaret Burnett, and George Robertson. 2009. 'Design Implications for End-User Debugging Tools: A Strategy-Based View'. <https://ir.library.oregonstate.edu/xmlui/handle/1957/12443>.
- Gugerty, L., and G. Olson. 1986. 'Debugging by Skilled and Novice Programmers'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 171–74. CHI '86. New York, NY, USA: ACM. <https://doi.org/10.1145/22627.22367>.
- Hanington, Bruce, and Bella Martin. 2012. *Universal Methods of Design: 100 Ways to Research Complex Problems, Develop Innovative Ideas, and Design Effective Solutions*. Beverly, MA: Rockport.
- Harrower, Mark, and Cynthia A. Brewer. 2003. 'Colorbrewer.Org: An Online Tool for Selecting Colour Schemes for Maps'. *The Cartographic Journal* 40 (1): 27–37. <https://doi.org/10.1179/000870403235002042>.
- Hartmann, Björn, Leith Abdulla, Manas Mittal, and Scott R. Klemmer. 2007. 'Authoring Sensor-Based Interactions by Demonstration with Direct Manipulation and Pattern Recognition'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 145–54. CHI '07. San Jose, California, USA: Association for Computing Machinery. <https://doi.org/10.1145/1240624.1240646>.
- Hartmann, Björn, Scott R. Klemmer, Michael Bernstein, Leith Abdulla, Brandon Burr, Avi Robinson-Mosher, and Jennifer Gee. 2006. 'Reflective Physical Prototyping through Integrated Design, Test, and Analysis'. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 299–308. UIST '06. New York, NY, USA: ACM. <https://doi.org/10.1145/1166253.1166300>.
- Hornecker, Eva. 2010. 'Creative Idea Exploration within the Structure of a Guiding Framework: The Card Brainstorming Game'. In , 101–8. ACM. <https://doi.org/10.1145/1709886.1709905>.
- 'IDEO Method Cards'. n.d. IDEO. Accessed 17 January 2018. <https://www.ideo.com/post/method-cards>.
- Igoe, Tom, and Dan O'Sullivan. 2004. *Physical Computing: Sensing and Controlling the Physical World with Computers*. First Printing edition. Boston: Premier Press.
- Inquirium, LLC. n.d. *InqScribe: Simple Software for Transcription and Subtitling*. <https://www.inqscribe.com>.
- Jayathirtha, Gayithri, Deborah A Fields, and Yasmin B Kafai. 2018. 'Computational Concepts, Practices, and Collaboration in High School Students' Debugging Electronic Textile Projects'. In *Proceedings of the International Conference on Computational Thinking Education 2018 (CTE 2018)*, 27–32. Hong Kong: The Education University of Hong Kong.

Jenkins, Tom, and Ian Bogost. 2014. 'Designing for the Internet of Things: Prototyping Material Interactions'. In *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, 731–40. CHI EA '14. New York, NY, USA: ACM. <https://doi.org/10.1145/2559206.2578879>.

Jernigan, W., A. Horvath, M. Lee, M. Burnett, T. Cui, S. Kuttal, A. Peters, I. Kwan, F. Bahmani, and A. Ko. 2015. 'A Principled Evaluation for a Principled Idea Garden'. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 235–43. <https://doi.org/10.1109/VLHCC.2015.7357222>.

Jernigan, William, Amber Horvath, Michael Lee, Margaret Burnett, Taylor Cui, Sandeep Kuttal, Anicia Peters, et al. 2017. 'General Principles for a Generalized Idea Garden'. *Journal of Visual Languages & Computing*, Special Issue on Programming and Modelling Tools, 39 (April): 51–65. <https://doi.org/10.1016/j.jvlc.2017.04.005>.

Jonassen, David H. 2000. 'Toward a Design Theory of Problem Solving'. *Educational Technology Research and Development* 48 (4): 63–85.

Jonassen, David H. 2010. *Learning to Solve Problems: A Handbook for Designing Problem-Solving Learning Environments*. New York: Routledge.

Jung, Malte F., Nik Martelaro, Halsey Hoster, and Clifford Nass. 2014. 'Participatory Materials: Having a Reflective Conversation with an Artifact in the Making'. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, 25–34. Vancouver BC Canada: ACM. <https://doi.org/10.1145/2598510.2598591>.

Kafai, Yasmin B., Eunkyong Lee, Kristin Searle, Deborah Fields, Eliot Kaplan, and Debora Lui. 2014. 'A Crafts-Oriented Approach to Computing in High School: Introducing Computational Concepts, Practices, and Perspectives with Electronic Textiles'. *Trans. Comput. Educ.* 14 (1): 1:1–1:20. <https://doi.org/10.1145/2576874>.

Katz, Irvin R., and John R. Anderson. 1987. 'Debugging: An Analysis of Bug-Location Strategies'. *Human-Computer Interaction* 3 (4): 351–99. https://doi.org/10.1207/s15327051hci0304_2.

Kelleher, Caitlin, and Randy Pausch. 2005. 'Lowering the Barriers to Programming: A Taxonomy of Programming Environments and Languages for Novice Programmers'. *ACM Comput. Surv.* 37 (2): 83–137. <https://doi.org/10.1145/1089733.1089734>.

Kissinger, Cory, Margaret Burnett, Simone Stumpf, Neeraja Subrahmaniyan, Laura Beckwith, Sherry Yang, and Mary Beth Rosson. 2006. 'Supporting End-User Debugging: What Do Users Want to Know?' In *Proceedings of the Working Conference on Advanced Visual Interfaces*, 135–42. AVI '06. New York, NY, USA: ACM. <https://doi.org/10.1145/1133265.1133293>.

Knörig, André, Reto Wettach, and Jonathan Cohen. 2009. 'Fritzing: A Tool for Advancing Electronic Prototyping for Designers'. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, 351–58. TEI '09. New York, NY, USA: ACM. <https://doi.org/10.1145/1517664.1517735>.

Ko, Amy J., Robin Abraham, Laura Beckwith, Alan Blackwell, Margaret M. Burnett, Martin Erwig, Chris Scaffidi, et al. 2011. 'The State of the Art in End-User Software Engineering'. *ACM Comput. Surv.* 43 (3): 21:1–21:44. <https://doi.org/10.1145/1922649.1922658>.

Ko, Amy J., Thomas D. LaToza, and Margaret M. Burnett. 2015. 'A Practical Guide to Controlled Experiments of Software Engineering Tools with Human Participants'. *Empirical Software Engineering* 20 (1): 110–41. <https://doi.org/10.1007/s10664-013-9279-3>.

- Ko, Amy J., and Brad A. Myers. 2003. 'Development and Evaluation of a Model of Programming Errors'. In *2003 IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings*, 7–14. <https://doi.org/10.1109/HCC.2003.1260196>.
- Ko, Amy J., and Brad A. Myers. 2004. 'Designing the Whyline: A Debugging Interface for Asking Questions About Program Behavior'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 151–58. CHI '04. New York, NY, USA: ACM. <https://doi.org/10.1145/985692.985712>.
- Ko, Amy J., and Brad A. Myers. 2005. 'A Framework and Methodology for Studying the Causes of Software Errors in Programming Systems'. *Journal of Visual Languages & Computing* 16 (1–2): 41–84. <https://doi.org/10.1016/j.jvlc.2004.08.003>.
- Ko, Amy J., and Brad A. Myers. 2008. 'Debugging Reinvented: Asking and Answering Why and Why Not Questions about Program Behavior'. In *Proceedings of the 30th International Conference on Software Engineering*, 301–10. ICSE '08. New York, NY, USA: ACM. <https://doi.org/10.1145/1368088.1368130>.
- Ko, Amy J., Brad A. Myers, and Htet Htet Aung. 2004. 'Six Learning Barriers in End-User Programming Systems'. In *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing*, 199–206. VLHCC '04. Washington, DC, USA: IEEE Computer Society. <https://doi.org/10.1109/VLHCC.2004.47>.
- Krippendorff, Klaus. 2012. *Content Analysis: An Introduction To Its Methodology*. 3 edition. Los Angeles; London: Sage Publications, Inc.
- Kulesza, Todd, Margaret Burnett, Simone Stumpf, Weng-Keen Wong, Shubhomoy Das, Alex Groce, Amber Shinsel, Forrest Bice, and Kevin McIntosh. 2011. 'Where Are My Intelligent Assistant's Mistakes? A Systematic Testing Approach'. In *End-User Development*, edited by Maria Francesca Costabile, Yvonne Dittrich, Gerhard Fischer, and Antonio Piccinno, 171–86. Lecture Notes in Computer Science 6654. Springer Berlin Heidelberg. http://0-link.springer.com.wam.city.ac.uk/chapter/10.1007/978-3-642-21530-8_14.
- Kulesza, Todd, Weng-Keen Wong, Simone Stumpf, Stephen Perona, Rachel White, Margaret M. Burnett, Ian Oberst, and Amy J. Ko. 2009. 'Fixing the Program My Computer Learned: Barriers for End Users, Challenges for the Machine'. In *Proceedings of the 14th International Conference on Intelligent User Interfaces*, 187–96. IUI '09. New York, NY, USA: ACM. <https://doi.org/10.1145/1502650.1502678>.
- Kultima, Annakaisa, Johannes Niemelä, Janne Paavilainen, and Hannamari Saarenpää. 2008. 'Designing Game Idea Generation Games'. In *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, 137–44. Future Play '08. New York, NY, USA: ACM. <https://doi.org/10.1145/1496984.1497007>.
- Kuttal, Sandeep Kaur, Anita Sarma, and Gregg Rothmel. 2013. 'Debugging Support for End-User Mashup Programming'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1609–18. CHI '13. New York, NY, USA: ACM. <https://doi.org/10.1145/2470654.2466213>.
- Kuznetsov, Stacey, and Eric Paulos. 2010. 'Rise of the Expert Amateur: DIY Projects, Communities, and Cultures'. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, 295–304. NordiCHI '10. New York, NY, USA: ACM. <https://doi.org/10.1145/1868914.1868950>.
- Kuznetsov, Stacey, Alex S. Taylor, Tim Regan, Nicolas Villar, and Eric Paulos. 2012. 'At the Seams: DIYbio and Opportunities for HCI'. In *Proceedings of the Designing Interactive Systems Conference*, 258–67. DIS '12. New York, NY, USA: ACM. <https://doi.org/10.1145/2317956.2317997>.
- Lahtinen, Essi, Kirsti Ala-Mutka, and Hannu-Matti Järvinen. 2005. 'A Study of the Difficulties of Novice Programmers'. *SIGCSE Bull.* 37 (3): 14–18. <https://doi.org/10.1145/1151954.1067453>.

LaToza, Thomas D., and Brad A. Myers. 2010. 'On the Importance of Understanding the Strategies That Developers Use'. In *Proceedings of the 2010 ICSE Workshop on Cooperative and Human Aspects of Software Engineering*, 72–75. CHASE '10. New York, NY, USA: ACM. <https://doi.org/10.1145/1833310.1833322>.

Lau, Sam, Sruti Srinivasa Ragavan, Ken Milne, Titus Barik, and Advait Sarkar. 2021. 'TweakIt: Supporting End-User Programmers Who Transmogrify Code'. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, 1–12. Yokohama Japan: ACM. <https://doi.org/10.1145/3411764.3445265>.

Lazar, Jonathan, Jinjuan Heidi Feng, and Harry Hochheiser. 2017. *Research Methods in Human-Computer Interaction*. 2nd edition. Cambridge, MA: Morgan Kaufmann.

'Learn How to Use Tinkercad'. n.d. Tinkercad. Accessed 7 June 2021. [/learn/circuits](https://learn.circuits).

Lee, Johnny C., Daniel Avrahami, Scott E. Hudson, Jodi Forlizzi, Paul H. Dietz, and Darren Leigh. 2004. 'The Calder Toolkit: Wired and Wireless Components for Rapidly Prototyping Interactive Devices'. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, 167–75. DIS '04. New York, NY, USA: ACM. <https://doi.org/10.1145/1013115.1013139>.

'LEGO Mindstorms'. n.d. LEGO. Accessed 30 June 2016. <http://www.lego.com/en-us/mindstorms/?domainredir=mindstorms.lego.com>.

Lesgold, Alan, and Susanne Lajoie. 1991. 'Complex Problem Solving in Electronics'. In *Complex Problem Solving: Principles and Mechanisms*, edited by Robert J. Sternberg and Peter A. Frensch, 287–316. Hillsdale, N.J.: Lawrence Erlbaum Associates, Inc.

Lesgold, Alan, Susanne Lajoie, Marilyn Bunzo, and Gary Eggan. 1992. 'SHERLOCK: A Coached Practice Environment for an Electronics Troubleshooting Job'. In *Computer-Assisted Instruction and Intelligent Tutoring Systems: Shared Goals and Complementary Approaches*, edited by Jill H. Larkin and Ruth W. Chabay, 201–38. Technology in Education Series. Hillsdale, NJ, England: Lawrence Erlbaum Associates, Inc.

Lewis, James R. 2006. 'Sample Sizes for Usability Tests: Mostly Math, Not Magic'. *Interactions* 13 (6): 29–33.

Lieberman, Henry. 2001. *Your Wish Is My Command: Programming By Example*. 1st edition. San Francisco: Morgan Kaufmann.

Lieberman, Henry, Fabio Paternò, Markus Klann, Volker Wulf, Henry Lieberman, Fabio Paternò, and Volker Wulf. 2006. 'End-User Development: An Emerging Paradigm'. In *End User Development*, 1–8. Human-Computer Interaction Series 9. Springer Netherlands. http://0-link.springer.com.wam.city.ac.uk/chapter/10.1007/1-4020-5386-X_1.

Liégeois, Laurent, G'érard Chasseigne, Sophie Papin, and Etienne Mullet. 2003. 'Improving High School Students' Understanding of Potential Difference in Simple Electric Circuits'. *International Journal of Science Education* 25 (9): 1129–45. <https://doi.org/10.1080/0950069022000017324>.

Lo, Jo-Yu, Da-Yuan Huang, Tzu-Sheng Kuo, Chen-Kuo Sun, Jun Gong, Teddy Seyed, Xing-Dong Yang, and Bing-Yu Chen. 2019. 'AutoFritz: Autocomplete for Prototyping Virtual Breadboard Circuits'. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, 1–13. Glasgow Scotland Uk: ACM. <https://doi.org/10.1145/3290605.3300633>.

Lockton, Dan, David Harrison, Tim Holley, and Neville A. Stanton. 2009. 'Influencing Interaction: Development of the Design with Intent Method'. In *Proceedings of the 4th International Conference on Persuasive Technology*, 5. Claremont, CA, USA: ACM. <https://doi.org/10.1145/1541948.1541956>.

- Lucero, Andrés, and Juha Arrasvuori. 2010. 'PLEX Cards: A Source of Inspiration When Designing for Playfulness'. In *Proceedings of the 3rd International Conference on Fun and Games*, 28–37. New York, NY, USA: ACM. <https://doi.org/10.1145/1823818.1823821>.
- Ludwig, Thomas, Oliver Stickel, Alexander Boden, and Volkmar Pipek. 2014. 'Towards Sociable Technologies: An Empirical Study on Designing Appropriation Infrastructures for 3D Printing'. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, 835–44. DIS '14. New York, NY, USA: ACM. <https://doi.org/10.1145/2598510.2598528>.
- Luger, Ewa, Lachlan Urquhart, Tom Rodden, and Michael Golembewski. 2015. 'Playing the Legal Card: Using Ideation Cards to Raise Data Protection Issues within the Design Process'. In , 457–66. ACM. <https://doi.org/10.1145/2702123.2702142>.
- Mackay, James, and Paul Hobden. 2012. 'Using Circuit and Wiring Diagrams to Identify Students' Preconceived Ideas about Basic Electric Circuits'. *African Journal of Research in Mathematics, Science and Technology Education* 16 (2): 131–44. <https://doi.org/10.1080/10288457.2012.10740735>.
- 'Make': n.d. Make: DIY Projects and Ideas for Makers. <http://makezine.com/>.
- Martin, F., B. Mikhak, and B. Silverman. 2000. 'MetaCricket: A Designer's Kit for Making Computational Devices'. *IBM Systems Journal* 39 (3.4): 795–815. <https://doi.org/10.1147/sj.393.0795>.
- Mayer, Richard E. 1981. 'The Psychology of How Novices Learn Computer Programming'. *ACM Comput. Surv.* 13 (1): 121–41. <https://doi.org/10.1145/356835.356841>.
- Mayring, Philipp. 2001. 'Combination and Integration of Qualitative and Quantitative Analysis'. *Forum Qualitative Sozialforschung / Forum: Qualitative Social Research* 2 (1). <https://doi.org/10.17169/fqs-2.1.967>.
- McCauley, Renee, Sue Fitzgerald, Gary Lewandowski, Laurie Murphy, Beth Simon, Lynda Thomas, and Carol Zander. 2008. 'Debugging: A Review of the Literature from an Educational Perspective'. *Computer Science Education* 18 (2): 67–92.
- McDermott, Lillian C., and Peter S. Shaffer. 1992. 'Research as a Guide for Curriculum Development: An Example from Introductory Electricity. Part I: Investigation of Student Understanding'. *American Journal of Physics* 60 (11): 994–1003. <https://doi.org/10.1119/1.17003>.
- McGrath, Will, Daniel Drew, Jeremy Warner, Majeed Kazemitabaar, Mitchell Karchemsky, David Mellis, and Björn Hartmann. 2017. 'Bifröst: Visualizing and Checking Behavior of Embedded Systems across Hardware and Software'. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 299–310. Québec City QC Canada: ACM. <https://doi.org/10.1145/3126594.3126658>.
- Mellis, David A, Massimo Banzi, David Cuartielles, and Tom Igoe. 2007. 'Arduino: An Open Electronics Prototyping Platform'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing (Alt.Chi)*, 11.
- Mellis, David A., Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. 'Engaging Amateurs in the Design, Fabrication, and Assembly of Electronic Devices'. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, 1270–81. DIS '16. New York, NY, USA: ACM. <https://doi.org/10.1145/2901790.2901833>.
- Mertens, Donna M. 2017. *Mixed Methods Design in Evaluation*. 1st edition. Evaluation in Practice. Los Angeles: SAGE Publications, Inc.

Métioui, Abdeljalil, Claude Brassard, Jude Levasseur, and Michel Lavoie. 1996. 'The Persistence of Students' Unfounded Beliefs about Electrical Circuits: The Case of Ohm's Law'. *International Journal of Science Education* 18 (2): 193–212. <https://doi.org/10.1080/0950069960180205>.

Miettinen, Reijo. 2000. 'The Concept of Experiential Learning and John Dewey's Theory of Reflective Thought and Action'. *International Journal of Lifelong Education* 19 (1): 54–72. <https://doi.org/10.1080/026013700293458>.

Millner, Amon, and Edward Baafi. 2011. 'Modkit: Blending and Extending Approachable Platforms for Creating Computer Programs and Interactive Objects'. In *Proceedings of the 10th International Conference on Interaction Design and Children*, 250–53. IDC '11. New York, NY, USA: ACM. <https://doi.org/10.1145/1999030.1999074>.

'MIT App Inventor'. n.d. Accessed 22 January 2021. <https://appinventor.mit.edu/>.

Mora, Simone, Francesco Gianni, and Monica Divitini. 2017. 'Tiles: A Card-Based Ideation Toolkit for the Internet of Things'. In *Proceedings of the 2017 Conference on Designing Interactive Systems - DIS '17*, 587–98. New York, NY, USA: ACM. <https://doi.org/10.1145/3064663.3064699>.

Morgan, Norah, and Juliana Saxton. 1991. *Teaching, Questioning and Learning*. London; New York: Routledge.

Morris, Nancy M., and William B. Rouse. 1985. 'Review and Evaluation of Empirical Research in Troubleshooting'. *Human Factors: The Journal of the Human Factors and Ergonomics Society* 27 (5): 503–30. <https://doi.org/10.1177/001872088502700502>.

Morville, Peter, and Louis Rosenfeld. 2007. *Information Architecture for the World Wide Web*. 3rd ed. Farnham: O'Reilly. <http://proquest.safaribooksonline.com/?uiCode=aberdeen&xmlId=0596527349>.

Mota, Catarina. 2011. 'The Rise of Personal Fabrication'. In *Proceedings of the 8th ACM Conference on Creativity and Cognition*, 279–88. C&C '11. New York, NY, USA: ACM. <https://doi.org/10.1145/2069618.2069665>.

Mueller, Florian, Martin R. Gibbs, Frank Vetere, Darren Edge, Florian Mueller, Martin R. Gibbs, Frank Vetere, and Darren Edge. 2014. 'Supporting the Creative Game Design Process with Exertion Cards'. In *CHI '14: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2211–20. New York, NY, USA: ACM. <https://doi.org/10.1145/2556288.2557272>.

Murphy, Laurie, Gary Lewandowski, Renée McCauley, Beth Simon, Lynda Thomas, and Carol Zander. 2008. 'Debugging: The Good, the Bad, and the Quirky – a Qualitative Analysis of Novices' Strategies'. In *Proceedings of the 39th SIGCSE Technical Symposium on Computer Science Education*, 163–67. SIGCSE '08. New York, NY, USA: ACM. <https://doi.org/10.1145/1352135.1352191>.

Nanja, Murthi, and Curtis R. Cook. 1987. 'An Analysis of the On-Line Debugging Process'. In *Empirical Studies of Programmers: Second Workshop*, edited by Gary M. Olson, Sylvia Sheppard, and Elliot Soloway, 172–84. Empirical Studies of Programmers. Norwood, NJ, USA: Ablex Publishing Corp.

Nardi, Bonnie A. 1993. *A Small Matter of Programming: Perspectives on End User Computing*. Cambridge, MA, USA: MIT Press.

'National Instruments LabVIEW'. 2013. 2013. <http://www.ni.com/labview/>.

Newell, Allen, and Herbert A Simon. 1972. *Human Problem Solving*. Englewood Cliffs, N.J.: Prentice-Hall.

Nielsen, Jakob. 2001. 'First Rule of Usability? Don't Listen to Users'. Nielsen Norman Group. 4 August 2001. <https://www.nngroup.com/articles/first-rule-of-usability-dont-listen-to-users/>.

‘Noun Project’. n.d. Noun Project. Accessed 18 March 2019. <https://thenounproject.com>.

Oehlberg, Lora, Wesley Willett, and Wendy E. Mackay. 2015. ‘Patterns of Physical Design Remixing in Online Maker Communities’. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 639–48. CHI ’15. New York, NY, USA: ACM. <https://doi.org/10.1145/2702123.2702175>.

O’Kane, Aisling Ann, Amy Hurst, Gerrit Niezen, Nicolai Marquardt, Jon Bird, and Gregory Abowd. 2016. ‘Advances in DIY Health and Wellbeing’. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 3453–60. CHI EA ’16. New York, NY, USA: ACM. <https://doi.org/10.1145/2851581.2856467>.

Otter.ai Inc. n.d. *Otter.Ai - Voice Meeting Notes & Real-Time Transcription*. Accessed 25 March 2022. <https://otter.ai/>.

Pane, John F., and Brad A. Myers. 1996. ‘Usability Issues in the Design of Novice Programming Systems’. Technical Report CMU-CS-96-132. Pittsburgh, Pennsylvania, USA: Carnegie Mellon University Institute for Software Research. <http://repository.cmu.edu/isr/820>.

Pane, John F., and Brad A. Myers. 2006. ‘More Natural Programming Languages and Environments’. In *End User Development*, edited by Henry Lieberman, Fabio Paternò, and Volker Wulf, 31–50. Human-Computer Interaction Series 9. Springer Netherlands. http://0-link.springer.com.wam.city.ac.uk/chapter/10.1007/1-4020-5386-X_3.

Pennington, Nancy, and Beatrice Grabowski. 1990. ‘The Tasks of Programming’. In *Psychology of Programming*, 45–62. London, UK: Academic Press. <https://www.cl.cam.ac.uk/teaching/1011/R201/ppig-book/ch1-3.pdf>.

Peppler, Kylie, and Diane Glosson. 2013. ‘Stitching Circuits: Learning About Circuitry Through E-Textile Materials’. *Journal of Science Education and Technology* 22 (5): 751–63. <https://doi.org/10.1007/s10956-012-9428-2>.

Periago, M. Cristina, and Xavier Bohigas. 2005. ‘A Study of Second-Year Engineering Students’ Alternative Conceptions about Electric Potential, Current Intensity and Ohm’s Law’. *European Journal of Engineering Education* 30 (1): 71–80. <https://doi.org/10.1080/03043790410001711225>.

Perkins, D. N., Chris Hancock, Renee Hobbs, Fay Martin, and Rebecca Simmons. 1986. ‘Conditions of Learning in Novice Programmers’. *Journal of Educational Computing Research* 2 (1): 37–55. <https://doi.org/10.2190/GUJT-JCBJ-Q6QU-Q9PL>.

Perkins, D. N., and Fay Martin. 1986. ‘Fragile Knowledge and Neglected Strategies in Novice Programmers’. In *Papers Presented at the First Workshop on Empirical Studies of Programmers on Empirical Studies of Programmers*, 213–29. Norwood, NJ, USA: Ablex Publishing Corp. <http://dl.acm.org/citation.cfm?id=21842.28896>.

Prabhakararao, Shrinu, Curtis Cook, Joseph R. Ruthruff, Eugene Creswick, Martin Main, Mike Durham, and Margaret M. Burnett. 2003. ‘Strategies and Behaviors of End-User Programmers with Interactive Fault Localization’. In *2003 IEEE Symposium on Human Centric Computing Languages and Environments, 2003. Proceedings*, 15–22. <https://doi.org/10.1109/HCC.2003.1260197>.

‘Processing’. n.d. Accessed 30 September 2012. <http://processing.org/>.

Reason, James. 1990. *Human Error*. Cambridge England ; New York: Cambridge University Press.

Repenning, Alexander, and Andri Ioannidou. 2006. ‘What Makes End-User Development Tick? 13 Design Guidelines’. In *End User Development*, edited by Henry Lieberman, Fabio Paternò, and Volker Wulf, 51–85. Human-Computer Interaction Series 9. Springer Netherlands. http://0-link.springer.com.wam.city.ac.uk/chapter/10.1007/1-4020-5386-X_4.

- Resnick, Mitchel, Robbie Berg, and Michael Eisenberg. 2000. 'Beyond Black Boxes: Bringing Transparency and Aesthetics Back to Scientific Investigation'. *Journal of the Learning Sciences* 9 (1): 7–30. https://doi.org/10.1207/s15327809jls0901_3.
- Resnick, Mitchel, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, et al. 2009. 'Scratch: Programming for All'. *Commun. ACM* 52 (11): 60–67. <https://doi.org/10.1145/1592761.1592779>.
- Resnick, Mitchel, Fred Martin, Randy Sargent, and Brian Silverman. 1996. 'Programmable Bricks: Toys to Think With'. *IBM Systems Journal* 35 (3.4): 443–52. <https://doi.org/10.1147/sj.353.0443>.
- Robertson, T. J., Shrinu Prabhakararao, Margaret Burnett, Curtis Cook, Joseph R. Ruthruff, Laura Beckwith, and Amit Phalgune. 2004. 'Impact of Interruption Style on End-User Debugging'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 287–94. CHI '04. New York, NY, USA: ACM. <https://doi.org/10.1145/985692.985729>.
- Rosson, Mary Beth, and John M. Carroll. 1993. 'Active Programming Strategies in Reuse'. In *ECOOP'93 — Object-Oriented Programming*, edited by Oscar M. Nierstrasz, 4–20. Lecture Notes in Computer Science 707. Springer Berlin Heidelberg. https://doi.org/10.1007/3-540-47910-4_2.
- Roy, Robin, and James Warren. 2018. 'Card-Based Tools for Creative and Systematic Design'. In *Proceedings of the Design Research Society DRS2018 Conference (TBC)*, 3:1075–87. Limerick, Republic of Ireland. <http://www.drs2018limerick.org/participation/proceedings>.
- Russo, J. Edward, Eric J. Johnson, and Debra L. Stephens. 1989. 'The Validity of Verbal Protocols'. *Memory & Cognition* 17 (6): 759–69. <https://doi.org/10.3758/BF03202637>.
- Ruthruff, Joseph R., Margaret Burnett, and Gregg Rothermel. 2005. 'An Empirical Study of Fault Localization for End-User Programmers'. In *Proceedings of the 27th International Conference on Software Engineering*, 352–61. ICSE '05. New York, NY, USA: ACM. <https://doi.org/10.1145/1062455.1062523>.
- 'S4A: Scratch for Arduino'. n.d. Accessed 18 March 2013. <http://seaside.citilab.eu/scratch/arduino>.
- Sadler, Joel, Lauren Shluzas, and Paulo Blikstein. 2017. 'Building Blocks in Creative Computing: Modularity Increases the Probability of Prototyping Novel Ideas'. *International Journal of Design Creativity and Innovation* 5 (3–4): 168–84. <https://doi.org/10.1080/21650349.2015.1136796>.
- Sarkar, A., M. Jamnik, A. F. Blackwell, and M. Spott. 2015. 'Interactive Visual Machine Learning in Spreadsheets'. In *2015 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 159–63. <https://doi.org/10.1109/VLHCC.2015.7357211>.
- Shaffer, Peter S, and Lillian C McDermott. 1992. 'Research as a Guide for Curriculum Development: An Example from Introductory Electricity. Part II: Design of Instructional Strategies'. *American Journal of Physics* 60 (11): 1003–13. <https://doi.org/10.1119/1.16979>.
- Sharp, Helen, Jennifer Preece, and Yvonne Rogers. 2019. *Interaction Design: Beyond Human-Computer Interaction*. 5th edition. Indianapolis, IN: Wiley.
- Shipstone, David. 1984. 'A Study of Children's Understanding of Electricity in Simple DC Circuits'. *European Journal of Science Education* 6 (2): 185–98. <https://doi.org/10.1080/0140528840060208>.

Shipstone, David. 1988. 'Pupils' Understanding of Simple Electrical Circuits. Some Implications for Instruction'. *Physics Education* 23 (2): 92. <https://doi.org/10.1088/0031-9120/23/2/004>.

Shneiderman, Ben. 2003. 'Promoting Universal Usability with Multi-Layer Interface Design'. In *Proceedings of the 2003 Conference on Universal Usability*, 8. New York, NY, USA: Association for Computing Machinery. [https://dl.acm-org.wam.city.ac.uk/doi/10.1145/957205.957206](https://dl.acm.org.wam.city.ac.uk/doi/10.1145/957205.957206).

Spohrer, James C., and Elliot Soloway. 1986. 'Novice Mistakes: Are the Folk Wisdoms Correct?' *Commun. ACM* 29 (7): 624–32. <https://doi.org/10.1145/6138.6145>.

Srnka, Katharina J., and Sabine T. Koeszegi. 2007. 'From Words to Numbers: How to Transform Qualitative Data into Meaningful Quantitative Results'. *Schmalenbach Business Review* 59 (1): 29–57. <https://doi.org/10.1007/BF03396741>.

Steinberg, Linda S., and Drew H. Gitomer. 1996. 'Intelligent Tutoring and Assessment Built on an Understanding of a Technical Problem-Solving Task'. *Instructional Science* 24 (3): 223–58. <https://doi.org/10.1007/BF00119978>.

Subrahmaniyan, Neeraja, Laura Beckwith, Valentina Grigoreanu, Margaret Burnett, Susan Wiedenbeck, Vaishnavi Narayanan, Karin Bucht, Russell Drummond, and Xiaoli Fern. 2008. 'Testing vs. Code Inspection vs. What Else?: Male and Female End Users' Debugging Strategies'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 617–26. CHI '08. New York, NY, USA: ACM. <https://doi.org/10.1145/1357054.1357153>.

Tanenbaum, Joshua G., Amanda M. Williams, Audrey Desjardins, and Karen Tanenbaum. 2013. 'Democratizing Technology: Pleasure, Utility and Expressiveness in DIY and Maker Practice'. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2603–12. CHI '13. New York, NY, USA: ACM. <https://doi.org/10.1145/2470654.2481360>.

Taylor, Chris. 2010. 'Beginner Troubleshooting - SparkFun Electronics'. SparkFun. 29 November 2010. <https://www.sparkfun.com/tutorials/226>.

Teddlie, Charles, and Abbas Tashakkori. 2010. 'Overview of Contemporary Issues in Mixed Methods Research'. In *SAGE Handbook of Mixed Methods in Social & Behavioral Research*, by Abbas Tashakkori and Charles Teddlie, 1–42. 2455 Teller Road, Thousand Oaks California 91320 United States: SAGE Publications, Inc. <https://doi.org/10.4135/9781506335193.n1>.

Tetteroo, Daniel, Iris Soute, and Panos Markopoulos. 2013. 'Five Key Challenges in End-User Development for Tangible and Embodied Interaction'. In *Proceedings of the 15th ACM on International Conference on Multimodal Interaction*, 247–54. ICMI '13. New York, NY, USA: ACM. <https://doi.org/10.1145/2522848.2522887>.

'The Oblique Strategies'. n.d. <http://www.rtqe.net/ObliqueStrategies/>.

Tomal, Daniel R., and Aram S. Agajanian. 2014. *Electronic Troubleshooting*. 4th ed. McGraw-Hill Education. <https://www.accessengineeringlibrary.com/content/book/9780071819909>.

Tudor, Leslie Gayle, Michael J. Muller, Tom Dayton, and Robert W. Root. 1993. 'A Participatory Design Technique for High-Level Task Analysis, Critique, and Redesign: The CARD Method'. In *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, 37:295–99. SAGE Publications Sage CA: Los Angeles, CA.

VERBI Software GmbH. n.d. *MAXQDA Analytics Pro 2020*. Accessed 25 March 2022. <https://www.maxqda.com/>.

Vessey, Iris. 1985. 'Expertise in Debugging Computer Programs: A Process Analysis'. *International Journal of Man-Machine Studies* 23 (5): 459–94. [https://doi.org/10.1016/S0020-7373\(85\)80054-7](https://doi.org/10.1016/S0020-7373(85)80054-7).

Villar, Nicolas, James Scott, and Steve Hodges. 2011. 'Prototyping with Microsoft .Net Gadgeteer'. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, 377–80. TEI '11. New York, NY, USA: ACM. <https://doi.org/10.1145/1935701.1935790>.

Wakkary, Ron, Markus Lorenz Schilling, Matthew A. Dalton, Sabrina Hauser, Audrey Desjardins, Xiao Zhang, and Henry W.J. Lin. 2015. 'Tutorial Authorship and Hybrid Designers: The Joy (and Frustration) of DIY Tutorials'. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 609–18. CHI '15. New York, NY, USA: ACM. <https://doi.org/10.1145/2702123.2702550>.

Wang, Chiuan, Hsuan-Ming Yeh, Bryan Wang, Te-Yen Wu, Hsin-Ruey Tsai, Rong-Hao Liang, Yi-Ping Hung, and Mike Y. Chen. 2016. 'CircuitStack: Supporting Rapid Prototyping and Evolution of Electronic Circuits'. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 687–95. UIST '16. New York, NY, USA: ACM. <https://doi.org/10.1145/2984511.2984527>.

Wetzel, Richard, Tom Rodden, and Steve Benford. 2016. 'Developing Ideation Cards for Mixed Reality Game Design'. In *Proceedings of 1st International Joint Conference of DiGRA and FDG*, 175–211. Dundee, UK.

Wickelgren, Wayne A. 1977. *How to Solve Problems: Elements of a Theory of Problems and Problem Solving*. San Francisco, Calif: W H Freeman & Co.

Wilkes, Maurice V. 1985. *Memoirs of a Computer Pioneer*. First edition. Cambridge, MA, USA: The MIT Press.

Williams, Elliot. 2015. 'Embed with Elliot: There Is No Arduino "Language"'. *Hackaday* (blog). 28 July 2015. <http://hackaday.com/2015/07/28/embed-with-elliott-there-is-no-arduino-language/>.

Wolber, David. 2011. 'App Inventor and Real-World Motivation'. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education - SIGCSE '11*, 601. Dallas, TX, USA: ACM Press. <https://doi.org/10.1145/1953163.1953329>.

Wölfel, Christiane, and Timothy Merritt. 2013. 'Method Card Design Dimensions: A Survey of Card-Based Design Tools'. In *SpringerLink*, 479–86. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-642-40483-2_34.

Wu, Te-Yen, Jun Gong, Teddy Seyed, and Xing-Dong Yang. 2019. 'Proxino: Enabling Prototyping of Virtual Circuits with Physical Proxies'. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, 121–32. New Orleans LA USA: ACM. <https://doi.org/10.1145/3332165.3347938>.

Wu, Te-Yen, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Jun-You Liu, Yu-Chih Lin, and Mike Y. Chen. 2017. 'CurrentViz: Sensing and Visualizing Electric Current Flows of Breadboarded Circuits'. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 343–49. Québec City QC Canada: ACM. <https://doi.org/10.1145/3126594.3126646>.

Wu, Te-Yen, Bryan Wang, Jiun-Yu Lee, Hao-Ping Shen, Yu-Chian Wu, Yu-An Chen, Pin-Sung Ku, Ming-Wei Hsu, Yu-Chih Lin, and Mike Y. Chen. 2017. 'CircuitSense: Automatic Sensing of Physical Circuits and Generation of Virtual Circuits to Support Software Tools.' In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 311–19. Québec City QC Canada: ACM. <https://doi.org/10.1145/3126594.3126634>.