

UNIVERSITY OF CALGARY

Designing Interactive Behaviours for Smart Objects

by

David Ledo Maira

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE
DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

AUGUST, 2020

© David Ledo Maira, 2020

ABSTRACT

In this thesis, I propose methods for repurposing existing hardware and software to enable designers to create live interactive prototypes for smart interactive objects without the need to write code or create custom circuitry. The advent of ubiquitous computing brought the promise of interactive artifacts that integrate into our everyday lives. While this has led to a myriad of “smart objects”, the problem is that it is difficult for interaction designers to devise interactive behaviours for such objects. For example, how might an interaction designer prototype behaviours for a smart speaker? How can they go beyond voice responses and, for instance, animate lights to show that the speaker is listening, or searching for an answer on the web? Designers today face three challenges: (1) needing multiple expertise of designing behaviour, form, circuitry, and programming the functionality; (2) lacking software tools to author fine-tuned dynamic behaviours; and (3) needing closer-to-product representations to physically manipulate the prototype.

I overcome this gap through a method and two interactive systems. First, I propose a design metaphor: *Soul-Body Prototyping*, which suggests leveraging off-the-shelf mobile phones and watches to create smart object prototypes. By enclosing the mobile device (“soul”) into a physical enclosure (“body”), the designer can exploit the mobile device’s rich sensing, outputs, and internet connectivity. I then operationalize *Soul-Body Prototyping* through two proof-of-concept prototyping tools. First, Pineal features trigger-action behaviours which automatically generate 3D models for physical forms. These forms fit a mobile device and expose the necessary inputs and outputs. Next, I present Astral, a tool in which designers can mirror a portion of the desktop’s screen onto a mobile device, and create simple rules that convert live mobile sensor data into mouse or keyboard events. Thus, the mobile device remote controls (and repurposes) familiar desktop applications in the context of dynamic behaviour prototyping.

Overall, my work contributes an alternative way to prototype smart interactive objects which informs the design of future prototyping tools. Moreover, I investigate fundamental questions such

as the meaning of interactive behaviour, and how to evaluate prototyping tools and toolkits in HCI research.

PUBLICATIONS

Many of the materials, figures and ideas presented in this dissertation have appeared in prior peer-reviewed publications.

Conference Publications

Hung, M., Ledo, D., & Oehlberg, L. (2019). WatchPen: Using Cross-Device Interaction Concepts to Augment Pen-Based Interaction. *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, 1–8. doi: [10.1145/3338286.3340122](https://doi.org/10.1145/3338286.3340122)

Ledo, D., Anderson, F., Schmidt, R., Oehlberg, L., Greenberg, S., & Grossman, T. (2017). Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2583–2593. doi: [10.1145/3025453.3025652](https://doi.org/10.1145/3025453.3025652)

Ledo, D., Houben, S., Vermeulen, J., Marquardt, N., Oehlberg, L., & Greenberg, S. (2018). Evaluation Strategies for HCI Toolkit Research. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–17. doi: [10.1145/3173574.3173610](https://doi.org/10.1145/3173574.3173610)

Ledo, D., Vermeulen, J., Carpendale, S., Greenberg, S., Oehlberg, L., & Boring, S. (2019). Astral: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications. *Proceedings of the 2019 on Designing Interactive Systems Conference*, 711–724. doi: [10.1145/3322276.3322329](https://doi.org/10.1145/3322276.3322329)

Extended Abstracts

Ledo, D. (2018). Designing Interactive Behaviours Beyond the Desktop. *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*, 228–231. doi: [10.1145/3266037.3266132](https://doi.org/10.1145/3266037.3266132)

Web Resources and Video Figures for Publications

The following web pages link to some of my projects at the time of this thesis submission (August 2020). These links contain information about specific projects, such as the article's PDF, video, and high-resolution images. The video figures are of particular importance to fully understand how the systems (WatchPen, Pineal and Astral) work in action.

Evaluation Strategies for HCI Toolkit Systems:

<http://davidledo.com/projects/project.html?toolkit-evaluation>

<https://github.com/davidledo/toolkit-evaluation>

WatchPen:

<http://davidledo.com/projects/project.html?watchpen>

Pineal:

<http://davidledo.com/projects/project.html?pineal>

Astral:

<http://davidledo.com/projects/project.html?astral>

THESIS CHAPTERS AND PUBLICATIONS

1 INTRODUCTION



Ledo, D. (2018). Designing Interactive Behaviours Beyond the Desktop. *The 31st Annual ACM Symposium on User Interface Software and Technology Adjunct Proceedings*, 228–231.

PART 1. PROTOTYPING INTERACTIVE BEHAVIOUR

2 BACKGROUND: INTERACTION DESIGN AND THE ROLE OF PROTOTYPING

3 DESCRIPTIVE FRAMEWORK OF INTERACTIVE BEHAVIOUR

4 PROTOTYPING INTERACTIVE BEHAVIOURS: RELATED WORK

5 EVALUATING TOOLKIT SYSTEMS



Ledo, D., Houben, S., Vermeulen, J., Marquardt, N., Oehlberg, L., & Greenberg, S. (2018). Evaluation Strategies for HCI Toolkit Research. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–17.

THEESIS CHAPTERS AND PUBLICATIONS

PART 2. SOUL-BODY PROTOTYPING

6 SOUL-BODY PROTOTYPING

7 CASE STUDIES: STUDENT EXPLORATIONS



Hung, M., Ledo, D., & Oehlberg, L. (2019). WatchPen: Using Cross-Device Interaction Concepts to Augment Pen-Based Interaction. *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, 1–8.

PART 3. SYSTEMS

8 PINEAL: PROTOTYPING FORM AND BEHAVIOUR IN TANDEM

9 ASTRAL: PROTOTYPING INTERACTIVE BEHAVIOUR WITH FAMILIAR TOOLS



ledo, D., Anderson, F., Schmidt, R., Oehlberg, L., Greenberg, S., & Grossman, T. (2017). Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2583–2593.

ledo, D., Vermeulen, J., Carpendale, S., Greenberg, S., Oehlberg, L., & Boring, S. (2019). Astralv: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications. *Proceedings of the 2019 on Designing Interactive Systems Conference*, 711–724.

10 CONCLUSIONS

TABLE OF CONTENTS

Abstract	i
Publications.....	iii
Table of Contents.....	iv
List of Figures	xv
Chapter 1. Introduction	1
1.1 Motivation: The Vision of Seamless Interactions with Technology.....	4
1.2 Problem: The Gaps When Designing a Single Smart Object.....	6
1.2.1 Challenge 1: Need for multiple specialization.	8
1.2.2 Challenge 2. Lack of Tool Support.....	9
1.2.3 Challenge 3. Need for Close-to-Product Representations.....	10
1.3 Thesis Statement	11
1.4 Research Questions.....	13
1.4.1 RQ1. How might designers repurpose mobile devices to prototype smart interactive objects?	14
1.4.2 RQ2. How might designers author forms around mobile devices to make them look and feel like smart objects?	15
1.4.3 RQ3. How might designers leverage existing familiar software tools to author interactive behaviours for smart objects?.....	15
1.4.4 Research Question Foundations.....	16
Who are interaction designers?	16
What is interactive behaviour?	17
How do we, or should we, evaluate prototyping tools?	17

1.5 Thesis Organization	18
1.5.1. Part 1: Prototyping Interactive Behaviour	18
1.5.2. Part 2: Soul-Body Prototyping.....	19
1.5.3. Part 3: Systems.....	20
PART 1 INTERACTIVE BEHAVIOURS	22
Chapter 2. Background	23
2.1 Interaction Design.....	25
2.1.1 What is Design?	25
2.1.2 User Experience Design	26
2.1.3 Interaction Design	27
2.1.4 Who are interaction designers?	28
2.1.5 How Designers Think	29
The Design Thinking Process.....	29
Strategic Aspects of Design Thinking.....	30
2.1.6 The Design Process: How Interaction Designers Work	32
The Product Development Process	32
The Design Phase	34
2.1.7 Summary	37
2.2 Prototyping: How Designers Explore Ideas	38
2.2.1 What is Prototyping?	38
2.2.2 Why Do Designers Prototype?.....	39
Prototyping as Exploration	39
Prototyping as Specification and Communication	40

Prototyping as Means of Evaluation	41
2.2.3 What do Prototypes Prototype?	42
2.2.4 Exploratory Prototyping in Interaction Design	44
Structure: The What	45
Behaviour: The How	45
Usage: First-Hand Experience with the Concept.....	46
2.2.5 Summary.....	46
Chapter 3. A Descriptive Framework of Interactive Behaviour	49
3.1 The Problem With The Word Feel	50
3.2 Interactive Behaviour, Defined	51
3.3 A descriptive Framework of Interactive Behaviour	53
3.3.1 Behaviours are Relationships Between Inputs and Outputs	54
3.3.2 Behaviours Have Dependencies	56
3.3.3 Programming Constructs of Behaviour.....	60
3.3.4 Summary	61
3.4 Conclusions	62
Chapter 4. Prototyping Interactive Behaviour and Related Work	63
4.1 Common Prototyping Activities.....	64
4.1.1 Sketching.....	64
4.1.2 Wireframes / Storyboards	65
4.1.3 Wizard of Oz	66
4.1.4 Video Prototyping	68
4.1.5 Programmed Interactive Prototypes	68

4.1.6 How Do These Approaches Prototype Structure, Behaviour and Usage?	70
4.2 How Do Designers Prototype Interactive Behaviour Today? Or do They?.....	71
4.2.1 Formative Interviews: Challenges and Needs.....	73
4.2.2 The (Commercial) Tools Designers Use Today	74
Software Tools Shape How People Think, And What Is Possible.....	76
4.3 Behaviour Prototyping Tools in Research and Industry	77
4.3.1 Traditional Coding	78
4.3.2 Visual Programming.....	80
4.3.3 Screen Transitions	82
4.3.4 Timeline	85
4.3.5 Programming by Example	86
4.3.6 Keyframing	88
4.3.7 Stage Metaphor	89
4.3.8 Step by Step Wizards	89
4.3.9 Wizard of Oz and Video Prototyping Tools	90
4.3.10 Smoke and Mirrors and Screen Poking.....	91
4.4 Summary And Conclusion.....	93
Chapter 5. Evaluation Methods	95
5.1 The Challenge of Toolkit Evaluation.....	96
5.2 What is a Toolkit?	98
5.2.1 Defining a Toolkit	98
5.2.2 Why do HCI Researchers Build Toolkits?	99
5.2.3 Evaluating Toolkits	103

5.3 Methodology.....	104
5.3.1 Dataset	104
5.3.2 Analysis and Results.....	105
5.4 Type 1: Demonstrations.....	106
5.4.1 Why Use Demonstrations?.....	107
5.4.2 Evaluation Techniques as Used in Demonstrations	108
5.4.3 Individual Instances	108
5.4.4 Collections	110
5.4.5 Going Beyond Descriptions.....	111
5.4.6 Challenges.....	112
5.4.7 Reflection and Opportunities	113
5.5 Type 2: Usage	114
5.5.1 Why Evaluate Usage?	115
5.5.2 Evaluation Techniques as Used in Usage Studies	116
5.5.3 Ways to Conduct Usage Studies.....	116
5.5.4 Ways to Elicit User Feedback	119
5.5.5 Challenges.....	121
5.5.6 Reflection and Opportunities	123
5.6 Type 3: Technical PErformance	124
5.6.1 Why Analyze the Technical Performance?	124
5.6.2 Techniques as Used in Technical Performance	124
5.6.3 Challenges.....	126
5.6.4 Reflection and Opportunities	127

5.7 Type 4: Heuristics.....	128
5.7.1 Why Use Heuristics?	129
5.7.2 Evaluation Techniques for Heuristics	130
5.7.3 Challenges	131
5.7.4 Reflection and Opportunities	132
5.8 Discussion.....	133
5.8.1 Rethinking Evaluation	133
Evaluation by Demonstration?.....	134
Usability Studies (Still) Considered Harmful Some of the Time	134
Successful Evaluation versus Successful Toolkit	137
Non-Coding Toolkits.....	137
The Need for HCI Infrastructure Research	141
5.9 Limitations.....	143
5.10 conclusion	143
PART 2 SOUL-BODY PROTOTYPING	144
Chapter 6. Soul-Body Prototyping.....	145
6.1 Motivation: Designer Challenges	146
6.2 Design Rationale	150
6.3 Soul-Body Prototyping Paradigm: The Mobile Device as a PPrototyping Engine.....	153
6.4 Design Space of soul-Body Prototyping.....	156
6.4.1 Mobile Sensors.....	158
6.4.2 Mobile Output	163
6.4.3 Power and Connectivity	163

6.4.4 Input and Output Modification.....	164
Rerouting Modifiers	164
Transducing Modifiers	165
6.5 Conclusion	166
Chapter 7. Soul-Body Prototyping Case Studies.....	169
7.1 Early Student Explorations.....	171
7.1.1 GymBuddy by Mike Chubey	172
7.1.2 Pathologist Device by Terrance Mok	173
7.1.3 Smart Docks by Orkhan Suleymanov	174
7.1.4 Huggable Phone by Sara Williamson	176
7.1.5 Phoame Swords by Kevin TA	176
7.2 Watchpen: Later Student Exploration	178
7.3 Discussion.....	181
7.3.1 Limitations	181
What can we extrapolate from student explorations?.....	181
Understanding Sensors.....	182
7.3.2 Soul-Body Prototyping and HCI Research.....	183
7.3.3 Soul Body Prototyping and Designers	184
7.4 Conclusion	185
PART 3 SYSTEMS	187
Chapter 8. Pineal: Behaviour-Driven Physical Prototyping	189
8.1 Pineal	190
8.2 Related Work and Contributions	193

8.3 Interface And Workspaces	196
8.3.1 Visual Programming Environment	196
Input Modules	196
Output Modules	198
Mapping Modules.....	199
8.3.2 Modeling Environment Interaction.....	200
Hover.....	201
Brush Selection.....	201
Object Placement.....	201
8.4 Usage Scenario: Creating a Toy Firetruck	202
8.4.1 Step 1: Importing the Base Form 3D Model	203
8.4.2 Step 2: Authoring the Behaviour via Visual Programming	203
8.4.3 Step 3: Guiding the Modelling Process	205
8.4.4 Step 4: Object Generation.....	206
8.4.5 Step 5: Object Assembly.....	207
8.5 Implementation.....	208
8.5.1 System Overview.....	208
8.5.2 Automated Model Configuration	211
Defining the Mobile Device Screen Position on the Form and Placing the Mobile Device	212
Screen Cavity and Alignment Pins.....	212
Speaker and Sensor Holes	213
Button Creation	214

LEDs Simulated with Light Pipes	214
Diffusers	215
8.5.3 Raw Sensor View	215
8.6 Resulting Prototypes.....	216
8.6.1 Toy Fire Truck.....	216
8.6.2 Magic 8 Ball	217
8.6.3 Level	218
8.6.4 Ambient Display Planter	218
8.6.5 Voice-Activated Light-bulb	220
8.7 Discussion.....	220
8.7.1 Limitations	221
Conceptual Limitations	221
Technical Limitations.....	227
8.7.2 Pineal, Designers and Prototyping	229
Pineal, Expertise and Current Practices.....	229
Is Automation Good?.....	230
How Rapid is Pineal's Rapid Prototyping?	232
How Long Should Pineal Prototypes Live?	233
Generality	233
8.8 Conclusion.....	233
Chapter 9. Astral	235
9.1 Astral	237
9.2 Related work and Contributions	242

9.3 Working with ASTRAL	249
9.3.1 Mirroring Desktop Contents	249
9.3.2 Specifying Input Remapping through Rules	249
9.3.3 Merging Several Rules into Rulesets	255
9.3.4 Deciding When Rules are Triggered	255
9.3.5 Sensor Selector.....	257
9.4 Usage Scenario: Creating a Level.....	259
9.4.1 Preparing the Prototype: Illustrator and AfterEffects.....	260
9.4.2 Step 1: Starting Astral	262
9.4.3 Step 2: Sensor Selector	262
9.4.4 Step 3: Rule Editor	263
9.4.5 Step 4: Mapping Mouse Position to the AfterEffects Timeline	263
9.4.6 Step 5: Fine-Tuning through Easing Functions.....	264
9.5 Interactive Prototypes Made With Astral	265
9.5.1 Converting Video into Interaction-Driven Animation.....	265
Video-Based Prototyping	265
9.5.2 Converting Existing Desktop Inputs/Outputs into New Device-Specific Interactions	267
Authoring Open-Ended Interaction Techniques	267
Prototyping Multiple Alternatives	269
9.5.3 Bringing Sketches to Life.....	269
Iterative Prototyping at Multiple Resolutions	269
Authoring Smart Object Behaviours	271

9.6 Implementation.....	272
9.7 Discussion.....	275
9.7.1 Revisiting the Design Rationale	276
9.7.2 Evaluation Approach.....	280
9.7.3 Scale and Reappropriation of Tools.....	283
9.7.4 Implementation-Level Constraints.....	285
Implementation Bottlenecks	285
Device Relativism	287
Permanence	288
9.8 Summary	288
Chapter 10. Conclusion	291
10.1 Revisiting This Thesis' target Problems	291
10.2 Primary Contributions	296
10.3 Secondary Contributions.....	297
10.4 Future Work	298
10.4.1 Making Existing Objects into Smart Prototypes	298
10.4.2 Soul–Body Multi-Device Ecologies.....	300
10.4.3 Supporting Multiple PArameters	301
10.4.4 New Building Blocks for Interactive Behaviour Design.....	302
10.4.5 Reflections in Systems Research	303
10.5 Reflection	304
10.6 Closing Remarks	306

LIST OF FIGURES

FIGURE 1.1 SCOPE OF RESEARCH PRESENTED IN THIS THESIS	3
FIGURE 1.2 VISUAL SUMMARY OF THESIS STATEMENT	13
FIGURE 1.3 THESIS OVERVIEW SHOWING THIS DISSERTATION'S CHAPTERS AND PARTS.....	21
FIGURE 2.1 USER EXPERIENCE DESIGN AS AN OVERLAP OF CONTENT, FORM AND BEHAVIOUR. (ADAPTED FROM COOPER ET AL. (2014)).	26
FIGURE 2.2 SURVEY RESULTS ADAPTED FROM PRATT AND NUNES (2012). PARTICIPANTS WERE ASKED: WHAT IS THE PROFESSIONAL BACKGROUND THAT LED TO YOUR UX POSITION?	28
FIGURE 2.3 DESIGN STRATEGIES OF CREATIVE DESIGNERS AS DESCRIBED BY CROSS (2011). DIAGRAM ADAPTED FROM CROSS (2011, PP.78)	31
FIGURE 2.4 DIFFERENT EXAMPLES OF THE DESIGN PROCESS, AS EXPLAINED BY DIX ET AL. (2004), COOPER ET AL. (2014), GREENBERG (1996) AND GIBBONS (2016). DIAGRAMS ADAPTED FROM THE RESPECTIVE SOURCES.	33
FIGURE 2.5 GETTING THE DESIGN RIGHT VS. GETTING THE RIGHT DESIGN, PARAPHRASED FROM GREENBERG ET AL. (2011).....	34
FIGURE 2.6 LASEAU'S VIEW (1982) OF THE DESIGN PROCESS AS ELABORATION AND REDUCTION. DIAGRAM BASED ON ILLUSTRATION BY GREENBERG ET AL. (2011).....	35
FIGURE 2.7 PUGH'S DESCRIPTION OF THE DESIGN PROCESS AS A FUNNEL (1990) WITH MULTIPLE CYCLES OF ELABORATION AND REDUCTION THAT CONVERGE TOWARDS A SOLUTION. DIAGRAM BASED ON ILLUSTRATION BY GREENBERG ET AL. (2012).	36
FIGURE 2.8 BUXTON'S (2007) DISTINCTION BETWEEN SKETCHING AND PROTOTYPING. WHILE IDEAS IN DESIGN INDEED NATURALLY EVOLVE FROM EXPLORATION TO SPECIFICITY, THE DESIGN LITERATURE STILL LARGELY REFERS TO ALL OF THESE TECHNIQUES AS PROTOTYPING. FIGURE REPRODUCED FROM BUXTON (2007).	39
FIGURE 2.9 SCHEMATIC OF THIS THESIS' EXPLORATORY PROTOTYPING IN THE CONTEXT OF INTERACTION DESIGN. GIVEN EXISTING RESEARCH ON WHAT PROTOTYPES PROTOTYPE, I ARGUE THAT EXPLORATORY PROTOTYPING HAS THE COMPONENTS OF STRUCTURE, BEHAVIOUR AND USAGE. STRUCTURE IS THE BASIS (FORM OR LAYOUT) WHICH THE PERSON INTERACTS WITH. BEHAVIOUR IS WHAT THE SYSTEM DOES BEFORE, DURING OR AFTER THE INTERACTION. USAGE REFERS TO A PERSON'S ABILITY TO TRY OUT THE BEHAVIOURS GIVEN THE PROVIDED STRUCTURE.	44
FIGURE 2.10 EXPLORATORY PROTOTYPING IN THE CONTEXT OF PRIOR WORK WHICH DISCUSSES WHAT PROTOTYPES MIGHT PROTOTYPE. NOTE THAT IN THE CONTEXT OF INTERACTION DESIGN, REFLECTING SYSTEM-LEVEL IMPLEMENTATION IS NOT IN THE SCOPE OF THE PROTOTYPING PRACTICE.	45
FIGURE 3.1 BUXTON's (1990) THREE-STATE MODEL OF GRAPHICAL INPUT SHOWING THE THREE STATES: (0) OUT OF RANGE, (1) TRACKING, AND (2) DRAGGING. IT ALSO SHOWS EXAMPLE INPUTS OF MOUSE, STYLUS/PEN, AND TOUCH AND HOW THEY MIGHT TRANSITION BETWEEN EACH STATE. FIGURE BASED ON BUXTON (1990).	56
FIGURE 3.2 MACKINLAY ET AL. (1990) TAXONOMY AS EXEMPLIFIED ON A RADIO. IMAGE BASED ON BUXTON (2013).	59

FIGURE 4.1 EXAMPLE OF QUICK SKETCHES FOR A SINGLE IDEA: “HOW MIGHT TWO PHONES SHARE A FILE BY USING THEIR BUILT-IN SENSORS?” IN THIS CASE, THE SKETCH SHOWS BUMPING TWO DEVICES TOGETHER TO SHARE FILES. WHILE EACH SKETCH SHOWS THE SAME IDEA, THEY HAVE DIFFERENT LEVELS OF VISUAL DETAIL, NONE OF WHICH TOOK MORE THAN TWO MINUTES TO COMPLETE.....	64
FIGURE 4.2 EXAMPLE OF A WIREFRAME. THE IMAGE ON THE LEFT SHOWS THE VIEW OF A PROFILE, WHILE THE VIEW ON THE RIGHT SHOWS WHAT HAPPENS WHEN A USER CLICKS ON THE TOP LEFT HAMBURGER MENU.	65
FIGURE 4.3 WIZARD OF OZ. ADAPTED FROM GOULD ET AL. (1983).....	66
FIGURE 4.4 CONTRASTING DIFFERENT PROTOTYPING APPROACHES AND THE EXTENT TO WHICH THEY PROTOTYPE STRUCTURE, BEHAVIOUR AND USAGE ELEMENTS OF AN INTERACTIVE SYSTEM OR ARTIFACT. NOTE HOW INTERACTIVE PROGRAMMED PROTOTYPES ARE OF THE HIGHEST FIDELITY AND RESOLUTION YET THEY ALSO PROVIDE THE MOST COVERAGE FOR WHAT THE SYSTEM MIGHT BE LIKE.....	71
FIGURE 4.5 TOOLS INTERACTION DESIGNERS USE TODAY AS DESCRIBED BY MYERS (2008), SUBTRACTION.COM (2015) AND UXTOOLS.CO (2018). RESULTS REPRODUCED FROM THE RESPECTIVE SOURCES.	75
FIGURE 4.6 EVENT MODEL VS. INTERACTION MACHINE. BEAUDOUIN-LAFON (2004) EXEMPLIFIES THE CONTRAST BETWEEN (A) EVENT-DRIVEN PROGRAMMING AND (B) AN INTERACTION MACHINE TO IMPLEMENT A RUBBER-BAND SELECTION. A RUBBER-BAND SELECTION CONSISTS OF ALLOWING THE USER TO DRAW A FREE-FORM AREA AND SELECT THE CONTENTS INSIDE OF IT (E.G. SHAPES, TEXT, IMAGES). FIGURE RECREATED FROM BEAUDOUIN-LAFON (2004).	79
FIGURE 4.7 EXAMPLES OF VISUAL PROGRAMMING APPROACHES. (A) AND (B) SHOW NODE-LINK DIAGRAMS MAX/MSP, WHERE (A) PERFORMS AUDIO MAPPINGS, WHILE (B) MODIFIES A VIDEO FEED. (C) SHOWS SCRATCH, A BLOCK-BASED LANGUAGE. (A) AND (B) TAKEN FROM HTTPS://CYCLING74.COM/PRODUCTS/MAX/ , WHILE (C) IS TAKEN FROM HTTPS://WWW.AACE.ORG/REVIEW/PREPARE-FOR-FUN-SCRATCH-3-0-IS-COMING/	82
FIGURE 4.8 THE D.TOOLS INTERFACE (ANNOTATED). IMAGE REPRODUCED AND MODIFIED FROM HARTMANN (2009).....	84
FIGURE 4.9 HOW MOTION TWEENING WORKS. THE ANIMATOR DRAWS THE FIRST FRAME, AND APPLIES TRANSFORMATIONS TO THE LAST FRAME (E.G. CHANGING FILL, SIZE AND ORIENTATION). THE SYSTEM AUTOMATICALLY INTERPOLATES BETWEEN THE TWO FRAMES TO CREATE A CONTINUOUS ANIMATION.	86
FIGURE 4.10 SETUP FOR CHAMELEON BY FITZMAURICE (1993). SETUP SHOWS HOW A PORTABLE TV AUGMENTED WITH A BUTTON AND POSITION SENSORS IS RENDERING THE VIDEO-STREAMED IMAGES FROM A CAMERA POINTED AT A MONITOR WHICH RUNS THE SOFTWARE APPLICATIONS FOR MOBILE SPATIAL NAVIGATIONS. IMAGE REPRODUCED FROM FITZMAURICE (1993).....	91
FIGURE 5.1 CODE AND CONTRIBUTION AS DESCRIBED BY FOGARTY (2017). FIGURE ILLUSTRATES HOW SYSTEM CONTRIBUTIONS ARE DESCRIBED WITHIN HCI RESEARCH: STAND-ALONE NOVEL TECHNICAL CONTRIBUTION; A COMBINATION OF NOVEL AND KNOWN TECHNIQUES TO ACHIEVE NOVEL FUNCTIONALITY; USING KNOWN TECHNIQUES TO ACHIEVE NOVEL FUNCTIONALITY; OR ACHIEVING KNOWN FUNCTIONALITY WITH NOVEL TECHNIQUES. FIGURED ADAPTED FROM (FOGARTY, 2017).....	99
FIGURE 5.2 GAINES’ (1991) BRETAM MODEL OF FORECASTING INFORMATION SCIENCES AS DESCRIBED BY GREENBERG (2007). WITHIN IT, TOOLKITS FOSTER REPLICATION TO AID DESIGN EXPLORATION (ADAPTED FROM GREENBERG (2007)).....	101
FIGURE 5.3. SUMMARY OF THE ENTIRE DATASET FOR TOOLKIT EVALUATION. EACH ROW REPRESENTS THE DIFFERENT TOOLKIT GOALS AND THEIR (INTERPRETED) METHOD DISTRIBUTIONS, WHILE THE LAST ROW SHOWS THE ENTIRE DATA DISTRIBUTION.....	136

FIGURE 5.4. DISTRIBUTION OF EVALUATION STRATEGIES FOR NON-PROGRAMMING TOOLKITS. HIGHEST VALUE ON THE CHART IS 14 PAPERS.

..... 138

FIGURE 6.1. NINTENDO LABO FROM 2018 ENABLES PLAYERS TO INSERT THE NINTENDO SWITCH TABLET AND CONTROLLERS INTO CARDBOARD ENCLOSURES. DIFFERENT CONTROLLER SENSORS (E.G. THE IR CAMERA) ARE USED TO SENSE USER INPUT AS DEFINED BY THE ENCLOSURE..... 154

FIGURE 6.2. M5STACK DEVICE. THE IMAGE SHOWS THE M5STACK BASE MODULE FEATURING A 2 INCH X 2 INCH TOUCHSCREEN, 3 BUTTONS, USB PORT, AND A LEGO-COMPATIBLE PEG AT THE BOTTOM TO ATTACH OTHER MODULES. THE BOTTOM THREE IMAGES SHOW THE MODULES FOR BATTERY, PROTOTYPING AND GPS. IMAGE ADAPTED FROM [HTTP://WWW.M5STACK.COM/](http://WWW.M5STACK.COM/) 155

FIGURE 7.1. GYMBUDDY (BY MIKE CHUBEY) IS A MOBILE DEVICE WHICH CAN BE ATTACHED TO GYM EQUIPMENT (E.G. BENCH PRESS) AND PROVIDES TRAINING AND ASSISTANCE THROUGH THE DISPLAY. THE SYSTEM KEEPS TRACK OF REPETITIONS VIA A DISTANCE SENSOR.

..... 172

FIGURE 7.2. PATHOLOGIST DEVICE (BY TERRANCE MOK). THE DEVICE FEATURES A CALIPER POWERED BY A KNOB AND A SERVO MOTOR WHICH IS USED TO RECORD MEASUREMENTS ON DIFFERENT BODY FEATURES (E.G. A TATTOO OR A SCAR). THE APPLICATION ALLOWS PATHOLOGISTS TO SELECT DIFFERENT PARTS OF THE BODY TO ENTER THE RECORDED INFORMATION..... 174

FIGURE 7.3. SMART DOCKS BY ORKHAN SULEYMANOV PROVIDES PHYSICAL CONTROLS AT DIFFERENT LOCATIONS OF A HOME WITH SPECIALIZED USAGE CONTEXTS: A MUSIC STATION, AS WELL AS AN ALARM CLOCK. 175

FIGURE 7.4. HUGGABLE PHONE BY SARA WILLIAMSON PRESENTS A STUFFED ANIMAL HUGGING A PHONE. THE MOBILE DEVICE CAN BE USED TO VIDEO CALL A PARENT WHO IS CURRENTLY AWAY. 176

FIGURE 7.5. PHOAME SWORDS BY KEVIN TA IS AN AUGMENTED PHYSICAL GAME IN WHICH PLAYERS ENGAGE IN A SWORD FIGHT UNTIL THEY RUN OUT OF HEALTH. THE MOBILE DEVICE PLAYS SOUND EFFECTS AND KEEPS TRACK OF PLAYER'S HEALTH / HIT POINTS. 177

FIGURE 7.6. WATCHPEN IS A TABLET STYLUS THAT HAS BEEN AUGMENTED WITH A SMARTWATCH. WATCHPEN EXPLORES HOW DIFFERENT SENSORS AND OUTPUTS CAN AUGMENT TABLET INTERACTIONS IN THE CONTEXT OF A DRAWING APPLICATION. 178

FIGURE 7.7 PHYSICAL AIRBRUSH (A), AND ITS REPLICATION IN WATCHPEN (B) WHICH CAN CONTROL THE INK FLOW WITH THE WATCH'S TOUCHPAD. THE ORIENTATION OF THE PEN (C) CHANGES HOW THE PAINT IS SPREAD ON THE CANVAS. 179

FIGURE 7.8. THE WATCHPEN DISPLAY SHOWS THE CURRENT COLOUR, AS WELL AS HUE, SATURATION, AND BRIGHTNESS AND RADIUS SLIDERS WHICH CAN BE ADJUSTED ANYTIME. 179

FIGURE 7.9. STAMP TOOL IN WATCHPEN, TRIGGERED WHEN THE PEN IS HELD UPRIGHT. THE PEN TOOL CAN CAPTURE CONTENTS (A) AND SHOW IT ON THE DISPLAY, AND THEN PASTE COPIES ON THE CANVAS (B). 180

FIGURE 8.1. OVERVIEW OF PINEAL. 191

FIGURE 8.2. PINEAL CONSISTS OF (1) A SIMPLIFIED VISUAL PROGRAMMING LANGUAGE TO AUTHOR BASIC BEHAVIOURS FOR MOBILE DEVICES (LEFT); AND (2) A 3D MODELING ENVIRONMENT, WHICH ALLOWS DESIGNERS TO IMPORT CUSTOM MODELS THAT CAN BE 3D PRINTED (RIGHT). PINEAL'S PROGRAMMED BEHAVIOURS THEN AUTOMATICALLY MODIFY THE 3D MODELS SO THEY CAN INDEED HOUSE THE MOBILE DEVICE AND EXPOSE THE NECESSARY INPUTS AND OUTPUTS ONCE 3D PRINTED. THIS FIGURE SHOWS A TOY

FIRETRUCK MODEL, WHICH IS USED AS A RUNNING EXAMPLE, WHERE PRESSING A BUTTON MAKES THE TRUCK FLASH ITS LIGHTS AND PLAY A SIREN SOUND.....	197
FIGURE 8.3. INTERACTIONS IN THE 3D MODELING ENVIRONMENT ARE OF THREE TYPES: (A) HOVERING THE CURSOR SHOWS WHERE AN OPERATION WILL TAKE PLACE; (B) PRESSING DOWN THE LEFT BUTTON AND DRAGGING ALLOWS THE USER TO PAINT A SELECTION; AND (C) PLACING AN IMPORTED MODEL. OPERATIONS ARE ALWAYS RELATIVE TO THE SURFACE OF THE 3D MODEL MESH.....	201
FIGURE 8.4. COMPONENTS OF THE COMPLETE FIRETRUCK MODEL.....	202
FIGURE 8.5. SCHEMATIC OF VISUAL PROGRAM TO CREATE INTERACTIVE FIRETRUCK.....	205
FIGURE 8.6. STEPS TAKEN BY THE DESIGNER TO CUSTOMIZE THE FIRETRUCK 3D MODEL FROM THE MOMENT IT IS IMPORTED UNTIL ALL MODIFICATIONS HAVE TAKEN PLACE. THE FIGURE ALSO SHOWS THE OPERATION THAT THE SYSTEM PERFORMS IN THE BACKGROUND.....	206
FIGURE 8.7. SYSTEM ARCHITECTURE OF PINEAL, SHOWING CONNECTIONS BETWEEN SMART PHONE, WATCH, NODEJS SERVER, C# CLIENT AND MESHMIXER	208
FIGURE 8.8. VISUAL DESCRIPTION OF WINDOW TRANSPARENCY: THE 3D MODELING ENVIRONMENT IN PINEAL IS AN INSTANCE OF MESHMIXER THAT SITS BEHIND THE PINEAL WINDOW.....	208
FIGURE 8.9. OPERATIONS PERFORMED BY PINEAL ON THE 3D MODELS, WHICH INCLUDE (A) POSITIONING THE SCREENS; (B) CARVING THE SCREEN AND PLACING ALIGNMENT PINS; (C) SPEAKER AND OTHER SENSOR HOLES; (D) BUTTONS; (E) SIMULATED LED LIGHTS THROUGH LIGHT PIPES AND (F) LIGHT DIFFUSER.....	211
FIGURE 8.10. EXAMPLE OF THE OPTIONAL SPRING AROUND THE BUTTON AS DONE IN THE FIRETRUCK PROTOTYPE. THE SPRING IS SOFT AND ELASTIC AND REQUIRES THE USER TO APPLY AN ACTIVATION FORCE FOR THE BUTTON TO MAKE CONTACT WITH THE MOBILE DEVICE SCREEN.....	213
FIGURE 8.11. LIVE ACCELEROMETER DATA FROM THE MOBILE DEVICE AS VISUALIZED BY PINEAL. THE VISUALIZATION PLOTS THE RAW X, Y AND Z VALUES.....	216
FIGURE 8.12. SAMPLE OF SMART INTERACTIVE OBJECTS CREATED WITH PINEAL, WHICH INCLUDES: (A) TOY FIRETRUCK; (B) MAGIC 8 BALL; (C) LEVEL; (D) AMBIENT LIGHT PLANTER; AND (E) VOICE-ACTIVATED LIGHT BULB	217
FIGURE 9.1. ASTRAL ALLOWS DESIGNERS TO PROTOTYPE INTERACTIVE BEHAVIOURS BY (1) MIRRORING CONTENTS OF A DESKTOP REGION TO A MOBILE DEVICE, (2) STREAMING MOBILE SENSOR DATA TO THE DESKTOP, AND (3) REMAPPING THE SENSOR DATA INTO DESKTOP INPUT (E.G. MOUSE AND KEYBOARD EVENTS) ON A DESIGNER-CHOSEN DESKTOP APPLICATION.....	238
FIGURE 9.2. ASTRAL'S STEPS TO CONVERT A DESKTOP WEBSITE INTO A MOBILE GAME.....	239
FIGURE 9.3. EASINGS IN ANIMATION. A CIRCLE MOVES HORIZONTALLY OVER TIME (FROM TIME = 0 TO TIME = N). TICKS ON THE TIMELINE MARK THE POSITION OF KEYFRAMES.....	248
FIGURE 9.4. ASTRAL'S MAIN INTERFACE AS DISPLAYED ON A DESKTOP COMPUTER, SHOWING THE (A) MAIN VIEW, AND (B) THE RULE EDITING WINDOW, WHERE DESIGNERS CAN CREATE MAPPINGS FROM SENSOR DATA ONTO MOUSE AND KEYBOARD EVENTS.....	250
FIGURE 9.5. RULES IN ASTRAL TAKE A SENSOR VALUE, SUCH AS THE ACCELEROMETER Y-AXIS, AND MAP IT TO A DESKTOP INPUT SUCH AS A MOUSE POSITION.....	251

FIGURE 9.6. SENSOR VISUALIZATIONS CHANGE DEPENDING ON THE CURRENTLY SELECTED SENSOR, TO ENABLE MORE STRAIGHTFORWARD MAPPINGS. THE FIGURE SHOWS (A) COMPASS, (B) TOUCHSCREEN, AND (C) LIGHT SENSOR.....	252
FIGURE 9.7. MAPPING SENSOR INPUTS TO DISCRETE AND CONTINUOUS KEYBOARD/MOUSE INPUTS. FIGURE SHOWS (A) MAPPING TO A DISCRETE INPUT SUCH AS A KEY PRESS, AND (B) MAPPING TO A CONTINUOUS INPUT DESTINATION (E.G. MOUSE POSITION).	253
FIGURE 9.8. EASING FUNCTION OPTIONS IN ASTRAL BASED ON PENNER (2002). THESE EASING FUNCTIONS CAN ALSO BE INVERTED.	255
FIGURE 9.9. SENSOR SELECTOR. DESIGNERS CAN RECORD AND PLAYBACK A VIDEO WHICH PLOTS THE SENSOR DATA OF ALL DETECTED MOBILE DEVICE SENSORS. DESIGNERS CAN THEN SCRUB ON THE VIDEO OR THE VISUALIZATIONS TO SEE THE ASSOCIATED VIDEO FRAME TO THAT SENSOR VALUE. DESIGNERS CAN CREATE A SELECTION TO AUTOMATICALLY CREATE A RULE THAT USES THE RANGE OF VALUES FOR THAT SENSOR.	258
FIGURE 9.10. VISUAL DESCRIPTION FOR ASTRAL'S USAGE SCENARIO. IN THE SCENARIO, A DESIGNER AUTHORS A PROTOTYPE FOR AN INTERACTIVE LEVEL ON A MOBILE DEVICE BY REPURPOSING THE TIMELINE ON A VIDEO EDITING SOFTWARE.	261
FIGURE 9.11. VIDEO BASED PROTOTYPES IN ADDITION TO THE LEVEL PHONE APP USED IN THE SCENARIO. (A) SHOWS A COMPASS, WHILE (B) SHOWS A RE-IMPLEMENTATION OF ANDROID'S QUICK SETTINGS MENU, WHERE ONE CAN CHANGE THE PHONE BRIGHTNESS.....	265
FIGURE 9.12. TILT-TO-ZOOM PROTOTYPE. THIS FIGURE SHOWS HOW ASTRAL LEVERAGES CONDITIONALS TO CREATE AN INTERACTION THAT REQUIRES CONSIDERABLE CODING ONLY REPURPOSING A WEB BROWSER ON THE DESKTOP.	268
FIGURE 9.13. INPUT VARIATIONS IN FLAPPY BIRD ON A MOBILE DEVICE, USING DIFFERENT INPUTS: (A) TOUCHING THE SCREEN, (B) BLOWING ON THE MICROPHONE, AND (C) SHAKING THE DEVICE. THE DESIGNER ALTERNATES THESE BY (D) COVERING THE SCREEN'S LIGHT SENSOR.	269
FIGURE 9.14. PROTOTYPES IN LOW RESOLUTIONS. THE FIGURE SHOWS (A) A SKETCH RENDERED ON THE MOBILE WATCH, (B) A POWERPOINT MOCKUP, AND (C) AN IN-PROGRESS DRAWING IN ILLUSTRATOR WHICH CONTROLS THE MUSIC IN ITUNES.....	270
FIGURE 9.15. SMART OBJECT PROTOTYPES FEATURING (A) A MUG HOLDING A SMART WATCH AS A SMART SPEAKER, AND (B) A PHYSICAL PROTOTYPE TO THE INITIAL VIDEO-BASED LEVEL PRESENTED IN THE FIRST USAGE SCENARIO.	271
FIGURE 9.16. SUMMARY OF THE INTERACTIVE PROTOTYPES, WHAT THE ORIGINAL DESIGN ACTIVITY ENTAILS AND HOW ASTRAL BRIDGES TAKING NON-INTERACTIVE PROTOTYPES INTO PROTOTYPES THAT CAN BE TESTED ON THE TARGET DEVICE AND REFINED BEFORE IMPLEMENTATION EFFORTS.....	273
FIGURE 9.17. RULE PREVIEW ENHANCEMENTS WITH ADDITIONAL FEEDBACK AND AWARENESS CUES.	278
FIGURE 9.18. ALIGNMENT TOOLS IN ADOBE ILLUSTRATOR.	284
FIGURE 10.1. SKETCH SAMPLE SHOWING DIFFERENT WAYS OF AUGMENTING EXISTING OBJECTS WITH MOBILE DEVICES TO WORK AS SMART OBJECT PROTOTYPES.....	300

CHAPTER 1. INTRODUCTION

“The only true voyage of discovery... is not to go to new places, but to see through different eyes” —Marcel Proust

In this thesis, I¹ propose methods for repurposing existing hardware and software to enable designers to create live interactive prototypes for smart interactive objects without having to write code or create custom circuitry.

While there is a sharp increase in the amount of technologies available today given the presence of smart objects such as digital assistant speakers, and smart lights, interaction designers’ practices have not kept up with these new technologies. In particular, interaction designers are missing methods and tools that can enable them to create prototypes to learn, discover, and realize ideas while considering the many variables involved in how people interact with technology.

¹ While my thesis is highly collaborative in my co-authored publications, I use the pronoun “I” as (1) I am the primary researcher leading and conducting the majority of this work, and (2) this collective body of work is being described and integrated within my own research agenda.

Some of these variables range from devising the inputs and outputs, to contextualizing the actions within the physical form of the object, to considering animations that can provide appropriate feedback for a user's actions. These variables and many more come together with the goal of providing a delightful user experience while ensuring usability. As it will be described in this thesis, there is a need for new tools and methods for interaction designers that can keep up with the increasing demands of designing smart interactive objects. By smart interactive objects, I refer to computationally powered, consumer-level physical objects, such as digital assistant speakers, smart lights, other appliances or even toys.

In general terms, my thesis contributes to- and is scoped within- the field of Human–Computer Interaction (HCI), a sub-discipline of computer science strongly influenced by design given its multi-disciplinary approach. The research I carry out in this thesis, contextualized in **Figure 1.1**, is situated at the intersection of different subareas within the field of HCI. The areas of *ubiquitous computing (ubicomp)* and *tangible user interfaces* explore how computers and technology integrate into our everyday lives to support activity beyond the confines of a single screen, which provides the perspective and theme behind my work. In my research, I create *prototyping tools*, new systems that provide building blocks to help the design and authoring of new interactive prototypes. In addition, my thesis borrows concepts from the branch of HCI of *mobile sensing and output techniques*, which provide a suite of approaches to realize the new prototyping tools and methods which can help create new technology.

HUMAN-COMPUTER INTERACTION (HCI)

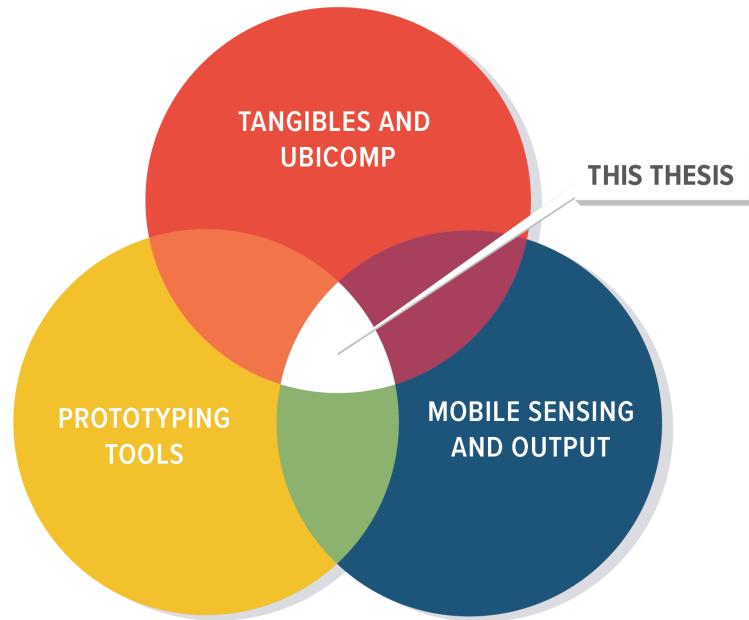


Figure 1.1 Scope of Research Presented in This Thesis

To enable interaction designers to create these new prototypes, it is first necessary to understand the gap between: the research vision proposed in Human–Computer Interaction (§1.1), and the challenges faced by interaction designers (§1.2). The careful consideration of these two realities will aid in the understanding of why *smart interactive objects* (e.g. digital assistant speakers, smart lights) might often fail to meet end-users' expectations, and why it is likely that interaction design is not yet formally incorporated in the creation process of smart objects. Thus, through the examination of the current state of the interaction design workflow, I propose my thesis statement as a solution to the design challenges in section §1.2, as specified through a design paradigm (Soul–Body Prototyping) and two systems (Pineal and Astra) (§1.3). To guide the execution of this thesis, I devise a series of research questions that I outline in section §1.4 and that guide my entire work, as well as a set of secondary questions which

establish the necessary foundation to understand and conduct the work presented in this dissertation. Finally, I discuss the organization of this thesis in §1.5.

1.1 MOTIVATION: THE VISION OF SEAMLESS INTERACTIONS WITH TECHNOLOGY

The field of ubiquitous computing stems from the vision that computers, and technology at large, will embed themselves into the fabric of everyday life (Weiser, 1990), where people interact with technology without realizing it. However, this embedding seems to be relative. For example, consider a person who lives in an urban setting using an elevator on a regular basis. This person might use it without noticing that there was an interactive exchange taking place with a computer, from the button presses, to the process of reaching the destination. For a technology to truly become invisible it means that it either is familiar enough to the point that the user has reached some form of mastery, or that the technology is “well designed”, meaning that it facilitates the person’s goals and people can make sense of their actions. It is possible to go beyond the purely functional focus, and think about the experience the object creates.

According to Norman (2013), interaction designers focus on ensuring that people can understand the technology that they operate, so they can know what can be done, what is happening and what just occurred when an action takes place. Cooper et al. (2014) distinguishes the focus on form from a graphic or industrial designer, with an interaction designer’s role in designing the interactive behaviour of an object to ultimately create a seamless user experience. As electronic compo-

nents become cheaper, new *smart interactive objects*, including appliances such as a smart lightbulb and a digital assistant speaker are becoming more common-place. These types of devices feature internet connectivity, and different means of interaction, such as light animations, sounds, buttons, or even touch-enabled displays. People together with different technologies can become integrated into an ecosystem, or an ecology (Nardi & O'Day, 1999), where the interactive objects are interconnected and support people's activities. The focus on many devices coming together has multiple flavours within the ubiquitous computing literature. Different explorations include computers assessing people's activity through sensing (Schilit et al., 1994), using that information to have technology react (Cooperstock et al., 1997), while taking into account people's location and social expectations (Greenberg, 2001). Indeed, examining a plethora of devices of different shapes and sizes with a wide range of functionality coming together shifts the focus of design towards human activity rather than dealing with a single device. The focus on human activity can be beneficial, as it helps reflect on how technologies can be reactive or proactive, and how interactions can take place in the foreground or background, so that technology can meet some of the social expectations (Ju & Leifer, 2008).

I argue, however, that people's operations with a single device are equally important, as they are often used individually to accomplish a particular task. Many of people's activities are mediated through interactions with a single device at a time. Thus, there is a need to carefully consider and refine how people interact with an individual object to create a seamless exchange. This way, people can come to understand what is possible to do with a smart object, as well as the

effect of their actions, all while having an experience that is free of pauses and hesitation, and even having one of delight.

One example of a non-seamless smart object interaction is a conversation with a smart speaker today (e.g. Google Home, Amazon Echo). Besides the expected delays and challenges associated with speech input, there are few ways for people to understand what is happening with the smart speaker. While the speaker may show some lights flashing or a delayed voice feedback, there is no way for the end-user to know if the speaker is listening to a question, or looking for answers on the internet, if there is an error, or what the confidence of the digital assistant is when providing an answer to the question. Here, an interaction designer can explore many aspects of how the exchange with the smart speaker takes place, such as showing an oscillating light that maps to the voice to show the response, animating the lights in a certain way to show that the speaker is listening, or looking for an answer, using colours to denote errors or low confidence, etc. The question then becomes how a designer can devise these types of rich interactions to help create a seamless and pleasurable experience with the smart object.

1.2 PROBLEM: THE GAPS WHEN DESIGNING A SINGLE SMART OBJECT

As mentioned, ubiquitous computing research tends to focus on human activity and how many devices can come together to support that activity. This does not mean that the design of the experience is compromised, rather it is secondary: the focus on what people are doing, and how they transition from one task to the next are augmented by

user experience elements. Indeed, one might still see ubiquitous computing work that considers aspects of animation and visuals, while also offering interesting mechanisms for interaction, as shown for instance in Proxemic Interaction work (Marquardt et al., 2011; Ledo et al., 2014). Similarly, single smart objects and their designs are also seen in HCI conferences, such as the Ripple Thermostat (van Oosterhout et al., 2018), which focuses on creating an emotional user experiences through force feedback and shape change. Still, it is necessary to first be able to design a single device interaction, and design it well, if we are to scale design to multiple devices working in concert. Moreover, many of the ideas presented in ubiquitous computing take a long time to be adopted, if at all, and it could largely be attributed to the lack of tools to create or simulate these experiences in the short term. As I discuss in Chapter 4, prototyping tools face cycles in which research and development increases, only to have new technologies arrive and clear some of the efforts as authoring ability becomes common-place again. However, the increases in standardization in the last decade have also led to additional barriers, as it is difficult to create prototypes that break away from the new standards.

The question then becomes, how to bring focus to these experiential elements, and how to further include interaction designers, who are trained in creating rich user experiences. Interaction design provides vocabulary and methods to devise how people interact with technology, and outline *interactive behaviours*: means for people to understand their actions and communication with interactive systems. For example, some behaviours might encompass suggesting how someone might interact with an artifact via: *affordances* and *signifiers* (e.g. a ‘button’ suggests something can be pressed), *mappings* (e.g. volume

increasing or decreasing as one turns a knob in a given direction), or providing appropriate *feedback* (e.g. showing the current temperature setting on a thermostat), etc. (Norman, 2013). Many of these aspects break down into what Saffer (2013) calls *microinteractions*, which he claims are ways to show an object has been designed with care.

As I will fully describe in Chapter 2, designers typically engage in the process of prototyping, where they iteratively generate various different solutions to one or more aspects to the problem, and in the process learn and discover a final solution. However, the gap to interaction designers participating in the design of interactive smart objects comes down to three main challenges, described below and further discussed in Chapter 6.

1.2.1 CHALLENGE 1: NEED FOR MULTIPLE SPECIALIZATION.

Prototyping enables exploring what is possible, as well as discovering new aspects to the solution. Yet, the design of interactive behaviour often depends on a physical form and base functionality being present. Realizing these prototypes for smart objects requires that interaction designers can to some extent generate: (1) the object's *physical form*, which provides the object its meaning, while also exposing the appropriate controls (inputs and outputs); and (2) the object's *functionality*, which requires programming, as well as creating the custom circuitry that connects the different sensors and outputs. Thus, the designer needs to consider many variables at once, each which can be time consuming, features a variety of specialized challenges, and overall are tailored towards experts in different areas. For example, a smart object prototype might require a designer to generate physical

forms either through some material or CAD software; create electronic circuitry that can be embedded into the form, which entails finding the appropriate components and soldering them in place; and then program and debug the prototype, all while guessing whether the bugs are a result of software issues or problems with assembling the electronic circuit itself (Booth et al., 2016). Because the prototyping process is about time-bound individual explorations, relying on different specialists would hinder the prototyping process by removing the designers' ability to discover and fine-tune elements by trial and error.

1.2.2 CHALLENGE 2. LACK OF TOOL SUPPORT

Most existing prototyping tools (e.g. InVision, Adobe XD) and resources for interaction design focus on building applications for desktop computing or basic mobile applications. Thus, a lot of the tools and techniques assume the traditional WIMP (Windows, Icon, Menu and Pointers) paradigm, which emphasizes a flow of interactions based on states and transitions (e.g. “when a button is pressed, switch to the next screen”). As it will be discussed in Chapter 3, such actions represent only a small fraction of interactive behaviours, and the nuanced communication one can have with an interactive system. Additionally, smart objects have a variety of ways in which people can interact with them, as they can often be physically held and manipulated, or rely on different sensors to interpret human actions (e.g. buttons, accelerometer, microphone). As a result, interaction design for smart objects can differ considerably from interaction design for desktop computers. Because existing design tools do not readily lend

themselves to designing rich and nuanced interactive behaviours, designers end up resorting to other applications to approximate some of their ideas. For example, a designer might work with a video editor to create an animation, and communicate the result with many descriptions for a developer (Maudet et al., 2017). Thus, having a broader range of tools would enable designers to create the appropriate representation depending on what kind of prototype they want to achieve.

1.2.3 CHALLENGE 3. NEED FOR CLOSE-TO-PRODUCT REPRESENTATIONS

While an interaction designer's explorations will vary in terms of *resolution*² (e.g. paper sketches versus an interactive prototype) across the process, holding a physical representation can greatly help contextualize the interactions with the physical object. For example, a small action such as shifting a paper sketch of a mobile interface into a phone already helps a designer understand aspects such as scaling issues when fitting many elements on a screen as shown by de Sá et al. (2008). Given that smart objects are each unique in shape, holding the physical object and testing the different button or sensor reactions create a dramatically different experience compared to a two-dimensional prototype. Common low-cost and “fast” approaches such as paper sketches, storyboards, or click-through slideshows cannot achieve the small subtleties of the interaction when prototyping, or

² The HCI literature often uses the word “*fidelity*” broadly to refer to both stage of the design and level of sophistication. As it will be explained later, Houde and Hill (1997) and many sources in design theory define *fidelity* as the stage in the design process, from “*resolution*”, which refers to the prototype’s degree of sophistication.

reflect a responsive experience. This is where Myers et al. (2008) distinguish that a majority of behaviours cannot be represented as simple state transitions which can be sketched or storyboarded. In the end, the design of the experience is directly tied to the designer creating the experience and making the necessary changes until the object “*feels right*”. Löwgren and Stolterman (2007) refer to this ability as a development of a designers’ judgement. Yet to arrive at a better judgement, it is important for designers to create a physical prototype that they can physically hold, manipulate, and change, where they can learn within the creation process. The creation process needs to be more *malleable*.

Given these challenges, the overarching research question that I identify and investigate in this thesis is:

How might we devise a means for designers to author interactive behaviours for smart interactive objects?

1.3 THESIS STATEMENT

To address the three challenges presented in previous section, I propose methods for repurposing existing technologies to work as base platforms for designing interactive behaviours. I posit using mobile devices in place of custom electronics, and leveraging existing desktop applications to author the design of interactive behaviours. Mobile devices are common place and contain a large number of sensors and outputs: high resolution touchscreens, speakers, microphones, cameras, accelerometers, gyroscopes, magnetometers, etc. Additionally, they are already built-in with batteries and the ability to connect to the internet. Consequently, designers can place mobile devices

that drive the computation and act as the “*soul*” of the smart object, while a fabricated enclosure can provide the object with a form, or “*body*”, designers then should be able to leverage or repurpose familiar desktop tools to author prototypes, either through augmenting and automating existing tools (e.g. aiding in the 3D modeling process) or by remapping the mobile sensor values in ways that can be recognized by the existing applications (e.g. streaming a desktop video and manipulating a video timeline as a phone moves). Based on the above points, my thesis statement is that:

We can repurpose existing hardware, such as mobile phones and watches, and software to enable designers to create live interactive prototypes for smart interactive objects without requiring code or custom circuitry.

Given my thesis statement, this leads to the following research points that guide my work:

1. **Paradigm.** I explore how designers can use mobile devices and their built-in sensors and outputs in place of electronics to author interactive smart object prototypes (*realized via a design metaphor: Soul-Body Prototyping*)
2. **Tools.** I create proof-of-concept prototyping tools that build on top of existing desktop applications to:
 - a. Create physical prototypes that enclose the mobile device in an appropriate form given interactive behaviour specifications (*realized via a system, Pineal*)

- b. Repurpose familiar desktop tools to author rich interactive behaviours that are driven by the user's interactions (*realized via a system, Astral*)

1.4 RESEARCH QUESTIONS

The research points in the last section, I propose one possible solution to enable designers to overcome the challenges of (1) needing multiple expertise, (2) lacking tool support, and (3) requiring close-to-product representations outlined in §1.2. Through a prototyping paradigm and two interactive systems, I provide methods in which designers can work with readily available technologies and bring them

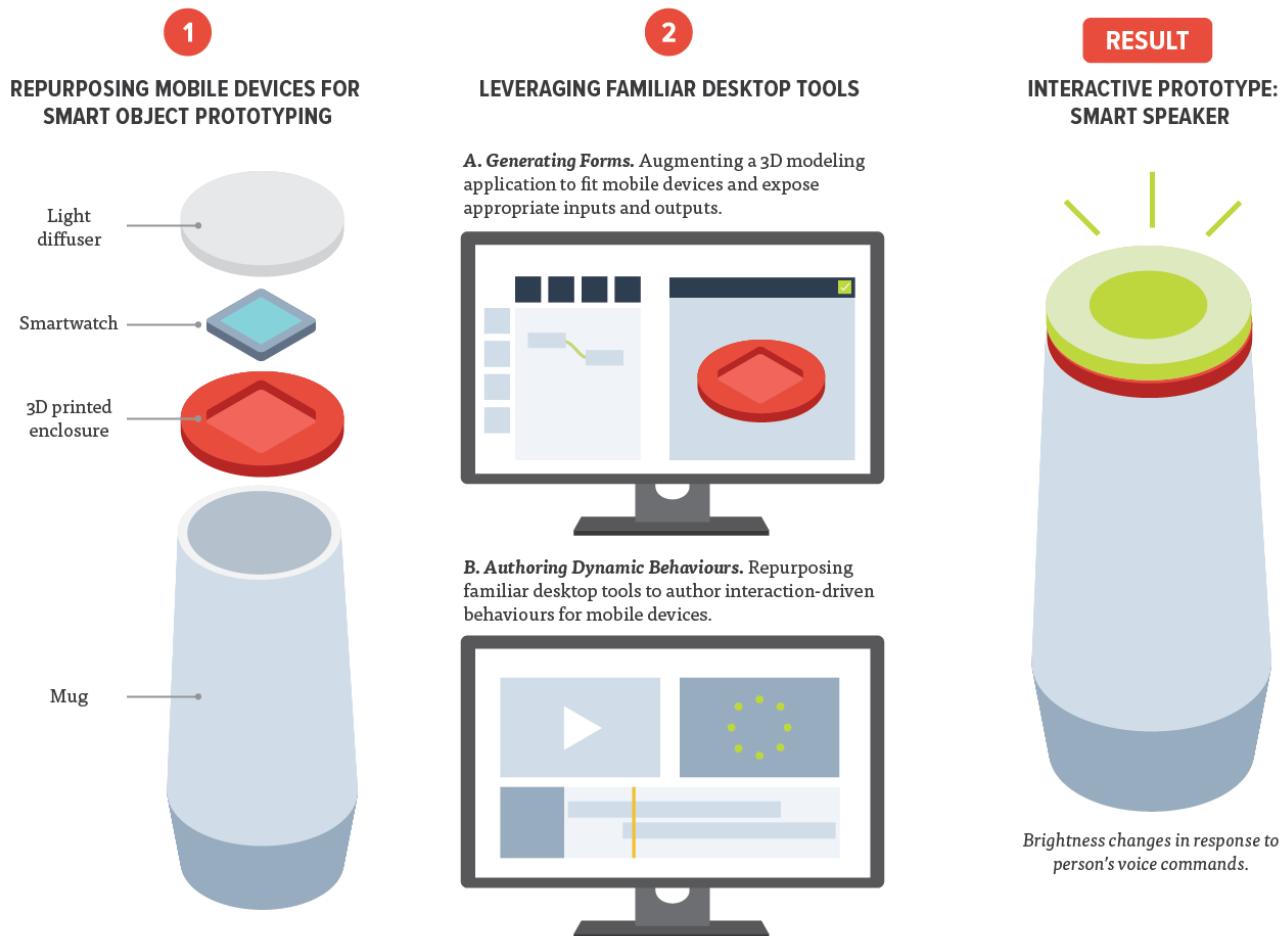


Figure 1.2 Visual Summary of Thesis Statement

into a new context. For example, a designer might create a prototype for a smart speaker (**Figure 1.2**) by placing a smart watch inside a mug, with the screen facing a lid that has a light diffuser, and may use tools to author the form for the lid containing the watch, as well as the watch's behaviours. The watch can then sense the speech input via the microphone, or when people hold the mug through the accelerometer, and flash a variety of animations. This approach proposes one way to answer the overarching research question in my work:

How might we devise means for designers to author interactive behaviours for smart interactive objects?

To realize the three research points within my thesis statement, I guide my work through three key research questions, summarized and contextualized within the smart speaker example in **Figure 1.2**.

1.4.1 RQ1. HOW MIGHT DESIGNERS REPURPOSE MOBILE DEVICES TO PROTOTYPE SMART ACTIVE OBJECTS?

To repurpose mobile devices as tools for smart object design, I propose Soul–Body Prototyping, a design metaphor in which the mobile device (*the ‘soul’*) is placed inside a physical form (*‘the body’*), which features the prototypes inputs and outputs and meaning. The paradigm enumerates the different sensors and outputs provided by mobile devices, and describe how they might be repurposed to create new kinds of inputs and outputs. Together with examples of how the sensing and outputs can be repurposed, it is possible for designers to appreciate what kinds of physical prototypes are possible, and how they might be devised. The added mobile device also provides a way

for designers to imagine and even invent new kinds of interesting smart objects. In an advanced HCI class, I provided students with an assignment in which they had to come up with new ways to use a mobile device. The variety of prototypes show the feasibility and the advantages of repurposing mobile devices rather than building smart objects from scratch. Answering this research question will address the first point in my thesis statement (**Figure 1.2-1**).

1.4.2 RQ2. HOW MIGHT DESIGNERS AUTHOR FORMS AROUND MOBILE DEVICES TO MAKE THEM LOOK AND FEEL LIKE SMART OBJECTS?

To generate physical forms in which the designer can incorporate the mobile device, I created Pineal. Pineal is a prototyping tool that can generate 3D printed physical objects that house mobile devices and expose the necessary inputs and outputs, informed by the Soul–Body Prototyping Paradigm (RQ1). Designers can import a pre-built 3D model, and use visual programming to author behaviours that run on the mobile device and also serve as instructions to modify the 3D model to fit a mobile device and use the appropriate inputs and outputs. This tool addresses point 2A of my thesis statement, as shown in **Figure 1.2-2A**.

1.4.3 RQ3. HOW MIGHT DESIGNERS LEVERAGE EXISTING FAMILIAR SOFTWARE TOOLS TO AUTHOR INTERACTIVE BEHAVIOURS FOR SMART OBJECTS?

I lastly explore how designers can create nuanced interactive behaviours through a prototyping tool called Astral. Astral allows designers

to use existing desktop tools and repurpose them for mobile device interactive behaviour design. It creates a closed-loop of interaction in which a portion of a desktop display is mirrored onto the mobile phone screen, and ranges of sensors are converted into mouse and keyboard events through interactive visualizations. Through Astral it is possible to use familiar desktop tools and encompass both the ability to trigger actions, as well as the ability to create interaction-driven animations, that is, animations that operate as a function of the interaction as opposed to simply a function of time. This last goal realizes point 2B of my thesis statement (**Figure 1.2-2B**).

1.4.4 RESEARCH QUESTION FOUNDATIONS

As I carried out my dissertation research, I found key points of knowledge to be missing within the field which were necessary to better understand the research at hand. As a result, the research questions are informed by, and sit upon integrating multiple theories in different fields beyond HCI, such as design research and interaction design. In particular, the following secondary questions arose:

Who are interaction designers?

Research in HCI often uses a broad term to talk about designers which does not necessarily reflect a specific design discipline. In particular, using the term “designer” in HCI often leads to a level of vagueness which allows researchers to stretch the definition. Zimmerman et al. (2007) have pointed out this problem of the need to distinguish interaction designers from software developers. As a result, it is necessary to best understand who interaction designers are, what their skillset is, and how they work if we are to design technologies to support their practice.

What is interactive behaviour?

The term “*interactive behaviour*” is often used to describe what interaction designers do (Cooper et al., 2014). While the term is effective at differentiating itself from form (i.e. physical form or layout of an interface), there is no definition of what is meant by interactive behaviour, therefore creating a fundamental ambiguity. While behaviour could refer to what an object does, there are many subtleties to behaviours that could be further clarified integrating different theories of input.

How do we, or should we, evaluate prototyping tools?

Hewett et al., (1992) define Human–Computer Interaction as a “*discipline concerned with the design, evaluation and implementation of interactive computing systems for human use*”. Indeed, Kaye (2007) points that knowledge creation requires some form of validation. As a result, HCI research often points to evaluation as a necessary component (Greenberg and Buxton, 2008), typically in the form of user studies, especially in the form of usability. However, the contribution of HCI systems research and evaluation in HCI research is often a controversial topic in the community given the variety of methods available to execute the research and the different perspectives coming to play. Having a solid foundation on evaluation methodologies helps strengthen the research creation process in two ways. First, it provides a framework to which one can assess the extent of the work done, and second, it provides a reference point of questions to consider when carrying out research even from the early stages.

Consequently, while these questions may seem secondary to the research I am carrying out, they are fundamental stepping stones to devise paradigms and tools to help interaction designers in the creation of smart objects.

1.5 THESIS ORGANIZATION

My thesis is structured in three main parts across ten chapters including this introduction, visually summarized in *Figure 1.3*.

1.5.1. PART 1: PROTOTYPING

INTERACTIVE BEHAVIOUR

The first part of the thesis focuses on the theory of *Interactive Behaviours* behind my work. The conceptual foundations in this part provide a base which defines (1) the target audience (interaction designers), (2) what interactive behaviours are, and (3) the contribution and role of evaluation of prototyping tools.

Chapter 2 explores and integrates different sources in HCI Research, Design Research, and Interaction Design Practitioner works and surveys. This integration helps define who interaction designers are, and what activities they carry out, which is something that often remains unclear in HCI literature. *Chapter 3* provides a descriptive framework of interactive behaviour, in which I bring together different approaches to create a working definition of behaviour in the context of interaction design. *Chapter 4* narrows down on prototyping approaches for interactive behaviour and shows the landscape of techniques and tools followed by practitioners today. These approaches and tools help to: understand the limitations of current tools, and situate my work within the landscape of tools that prototype different

degrees of prototyping interactivity. I further demonstrate that a lot of the expressiveness and fine-tuning of interactive behaviour is achieved through coding which is often beyond the expertise of designers. Then, I provide a taxonomy of prototyping tools in HCI and industry and show the different approaches and their processes. The question that arises from examining these tools is their research contribution and their evaluation, which provides further inquiry on the contribution of my tools. This is a larger challenge in Human–Computer Interaction Research, as prototyping tools fall into the category of HCI toolkits, which do not have well-established methods to evaluate the work. *Chapter 5* contextualizes the role and contribution of toolkits in HCI research, and analyzes the different evaluation methods through a survey of 68 representative papers. These results help further shape the meaning of evaluation in HCI systems research while also providing a set of methods that ultimately defined my own research approach in this thesis.

1.5.2. PART 2: SOUL–BODY PROTOTYPING

The second part of my thesis, *Soul–Body Prototyping* focuses on the primary conceptual contribution of my work. *Chapter 6* outlines *Soul–Body Prototyping* as a metaphor and paradigm for designers. In it, I suggest designers can create smart object prototypes by using mobile devices in place of custom electronics and placing them into forms that provide the prototype with meaning and physical inputs and outputs. I provide a design space of Soul–Body Prototyping which looks at ways to repurpose different mobile sensors and outputs. *Chapter 7* demonstrates Soul–Body Prototyping as a feasible and expressive paradigm by featuring and analyzing undergraduate

HCI student projects. I discuss a selection of five early explorations with undergraduate students in a five-week assignment, as well as a full Soul-Body research system, WatchPen, which represents a case study of a 3-month undergrad research project I supervised.

1.5.3. PART 3: SYSTEMS

The third part of the thesis focuses on *Systems* and is built upon the knowledge outlined in the previous part. It operationalizes Soul-Body Prototyping into software tools designers can use. *Chapter 8* shows *Pineal*, which supports designers in creating physical prototypes with basic behaviours through a 3D modeling tool and a visual programming environment. *Chapter 9* shows *Astral*, a prototyping tool which focuses on the creation of interactive behaviours for mobile devices and smart objects using existing familiar desktop tools.

Lastly, *Chapter 10* presents overall conclusions for my works, integrating the resulting research contributions from previous chapters and providing reflections on what this work means for existing practices. Here, I also discuss possible future directions for the continuation of this work.

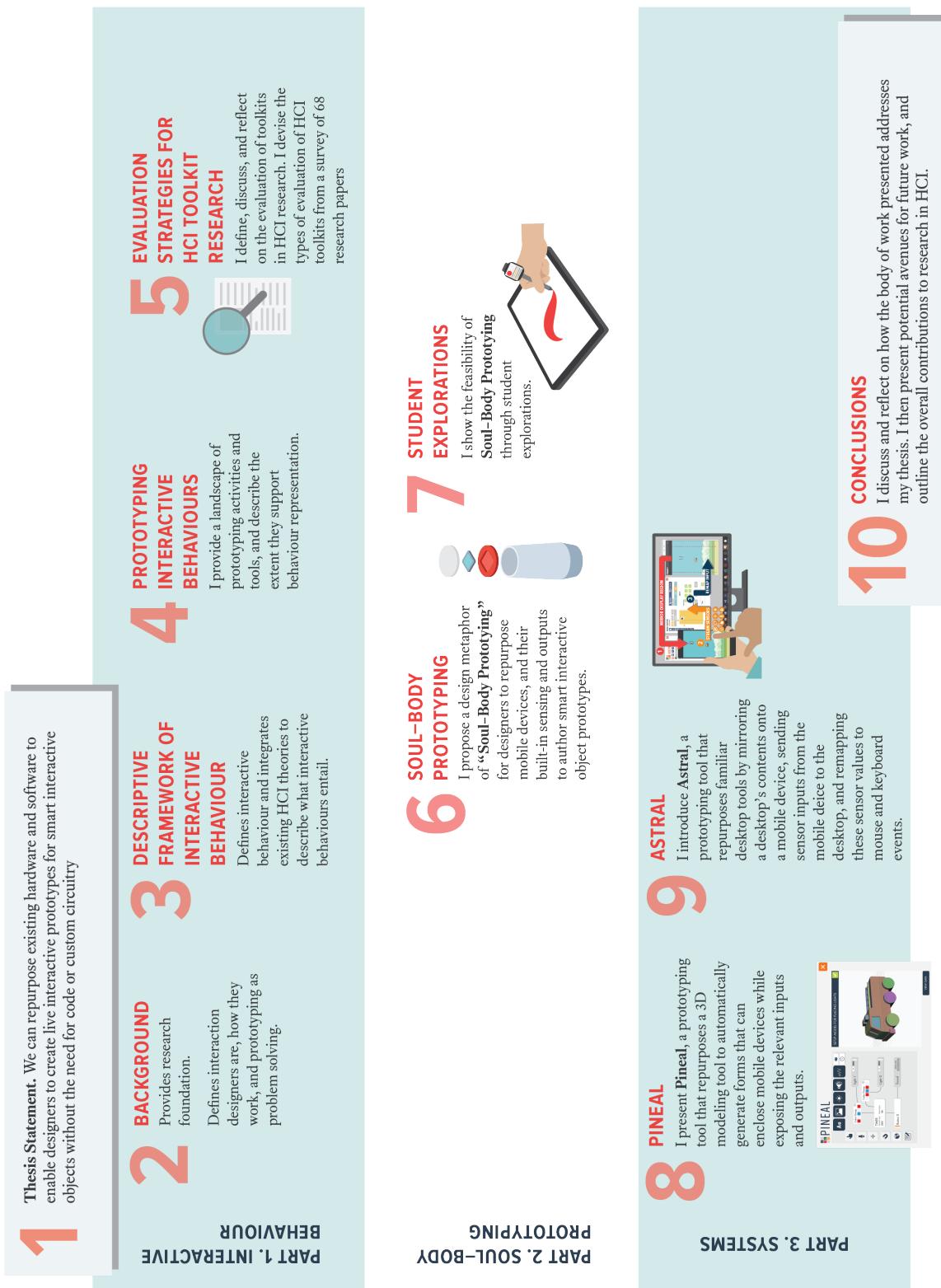


Figure 1.3 Thesis overview showing this dissertation’s chapters and parts.

PART 1

INTERACTIVE

BEHAVIOURS

CHAPTER 2. BACKGROUND

“Santayana taught us that those who do not know history are condemned to repeat it. That surely is true in design as in anything else, but in design there is a corollary: those who do know history are privileged to repeat it at a profit.” —Ralph Caplan

Since I propose methods for repurposing existing hardware and software to enable designers to prototype interactive behaviours for smart interactive objects, it is important to answer three foundational questions that bound this thesis: who interaction designers are; what interactive behaviours are; and how to assess the value of prototyping tools for interactive behaviour. In particular, this chapter has the added value of being an integration of different sources in what might seem as similar, yet are disconnected areas, including Design Research, Human-Computer Interaction, Interaction Design, and User Experience Design. Thus, this integration provides base definitions, as well as an

added understanding and structure to the role of interaction designers in designing interactive behaviours. To answer the three foundational questions, I structure this review as follows:

Interaction Designers (§2.1). One discipline that tends to be under-defined and sometimes misinterpreted in HCI is interaction designer. In particular, what needs further explanation is the kind of background an interaction designer has, how they think, and what their role is. To provide a holistic understanding of interaction design, I begin by defining and scoping what is meant by design (§2.1.1), user experience design (§2.1.2), and then narrow down into interaction design (§2.1.3). The thought process of interaction designers is similar to other design disciplines (§2.1.4), what changes is their role in the product development process (§2.1.5), and specifically how their ideas come together in the design phase (§2.1.6). Knowing these elements helps better understand the target audience of this thesis.

Prototyping (§2.2). Given that designers work through prototyping, understanding prototyping in the context of interaction design is a fundamental building block for this thesis. Interaction designers explore their ideas through prototypes. I explain what prototyping is (§2.2.1), why designers do it (§2.2.2) and what it might serve in the product design process (§2.2.3). I discuss how the literature interprets what types of questions prototypes can examine (§2.2.4) and frame interaction design prototypes into what I call *Exploratory Prototyping* (§2.2.5). Framing prototyping as exploratory prototyping puts together the different variables that prototypes might investigate and teases out interactive behaviour from that.

2.1 INTERACTION DESIGN

This section answers the first foundational question posed in this thesis of who interaction designers are. I discuss the background pertaining to interaction designers, and connect different sources to provide an understanding of the target audience for the systems and tools offered in this thesis. This definition requires narrowing down from the definition of Design (§2.1.1), to the more specific User Experience Design (§2.1.2), from which Interaction Design is a sub-discipline (§2.1.3). Once interaction design is defined, it is possible to discuss who interaction designers are (§2.1.4), and explain how designers are trained to think about problems generally (§2.1.5) before introducing the design process in larger product development as carried out in interaction design practice (§2.1.6).

2.1.1 WHAT IS DESIGN?

Before delving into the specifics of interaction design, I first explain what I mean by design. Design can refer to an individual **instance** (e.g. discussing a design with a client) or as a **process** (e.g. designing a chair). In the context of my thesis, I refer to design as a process.

The Oxford Dictionary defines design as the “*purpose, planning, or intention that exists or is thought to exist behind an action, fact, or material object*”. Dix et al. (2003) describe design as “*achieving goals within constraints*”, which articulates that design has a specific **purpose**, a set of **constraints** (e.g. time, budget, materials), and **trade-offs**. Caplan (1982) states that “*design is a process for making things right. For shaping what people need*”, which brings a key point – in achieving the specified purpose, design ultimately will reach people and has the

potential to impact their lives. Hartmann (2009) articulates that design has three core characteristics: (1) it is a **process** and has a **structure**, (2) it is not manufacturing or software development, and (3) it has **clients** and **end-users**.

Thus, I define design as:

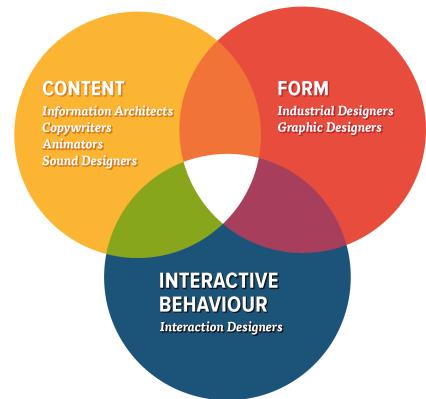
Design is the structured, and informed process for which things (actions or objects), which ultimately reach an audience, are intentionally made under specific constraints.

2.1.2 USER EXPERIENCE DESIGN

One discipline of design which has emerged in the last few decades, especially manufacturing and technologies become more commonplace, is the area of user experience design. Norman (2013) describes experience design as “*the practice of designing products, processes, services, events, and environments with a focus placed on the quality and enjoyment of the total experience*”. Cooper et al. (2014) claim that all realms of design influence people’s experiences by “*carefully manipulating the variables intrinsic to the medium at hand*” (pp. xxii). To show how different design disciplines craft experiences, they use different examples, such as: a graphic designer creating an experience through a poster by manipulating fonts, photos and illustrations; an industrial designer creating an experience through a chair by combining different materials and construction techniques; and an interior designer creating an experience for a space by using layout, lighting and materials.

Cooper et al. (2014) further refine the context of user experience design to the creation of digital products, which aligns with definitions

USER EXPERIENCE DESIGN (UX)



*Figure 2.1 User Experience Design as an overlap of content, form and behaviour.
(Adapted from Cooper et al. (2014)).*

by Norman (2013) and Moggridge (2007). Within user experience design, Cooper et al. (2014) promote three overlapping concerns: **form** (done by graphic and industrial designers), **content** (done by information architects, copywriters, animators, and sound designers), and **behaviours** (done by interaction designers), illustrated in **Figure 2.1**. The particular nuancing of these three concerns, as it will be described later, is important, as it helps separate the elements of layout and form from the elements of behaviour.

2.1.3 INTERACTION DESIGN

Cooper et al.'s (2014) depiction of user experience design portrays interaction design as a piece of the larger puzzle, which they explain is concerned specifically with the creation of "*interactive behaviour*". Löwgren and Stolterman (2007) define interaction design as "*the process that is arranged within existing resource constraints to create, shape, and decide all use-oriented qualities (structural, functional, ethical, and aesthetic) of a digital artifact*" (pp. 5). The Interaction Design Association (IxDA)¹ describes how the technology designed can encompass computers, mobile devices, appliances and beyond. Moreover, Norman (2013) brings forward the focus of interaction design to *how* people interact with technology, where "*the goal is to enhance people's understanding of what can be done, what is happening and what has just occurred. Interaction design draws upon principles of psychology, design, art, and emotion to ensure a positive, enjoyable experience*" (pp. 5).

¹ Interaction Design Association (IxDA) <https://ixda.org/ixda-global/about-history/> – accessed December 21, 2018.

2.1.4 WHO ARE INTERACTION DESIGNERS?

Given the multidisciplinary nature of interaction design, it does not yet have a well-defined home discipline compared to other forms of design such as architecture and graphic or industrial design. The discipline from where a practitioner comes from can shape their overall approach, as the formal training provides practitioners with: (1) tools and foundational theories (Stolterman, 2008), (2) methods and processes to approach problems (Cross, 1982), and (3) ways of knowing (Cross, 1982). An online survey by Pratt and Nunes (2012), illustrated in **Figure 2.2**, asked respondents about their formal background in user experience design, which showed that over 68% of designers are not formally trained in specialized technical disciplines (e.g. computer science). Other surveys (Subtraction.com 2015, Ux-tools.co 2017 and 2018), as well as prior studies in the HCI community, such as Myers et al. (2008) and Maudet et al. (2017) also discuss what tools interaction designers use, as well as their expertise, which corroborate this information and will be further described in Chapter 4, Section §4.2. Different design disciplines (e.g. graphic and industrial design) all have similar foundations as part of their training, though the specific activities carried out may vary. This training provides designers with a way of thinking different to ways taught in other disciplines (e.g. computational thinking²).

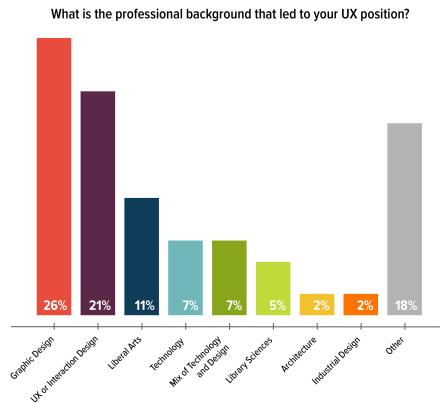


Figure 2.2 Survey results adapted from Pratt and Nunes (2012). Participants were asked: *What is the professional background that led to your UX position?*

² By “computational thinking”, I refer to the way of thinking taught in areas such as computer science, by which problems are decomposed into abstraction with the goal of automation (Wing, 2008).

2.1.5 HOW DESIGNERS THINK

“Designers in action are commonly described as being intuitive or sensitive to a situation. Sometimes the process is even seen as badly structured, subjective, or fuzzy. This same process can, however, also be seen as a highly rigorous and disciplined way to act if seen from a designerly point of view” (Stolterman, 2008).

Design problems are often described as “*wicked problems*” (Buchanan, 1992). According to Rittel and Webber (1973), wicked problems: have no definitive formulation and requires developing an exhaustive list of potential solutions; have no clear stopping rule, with multiple right and wrong answers; and with changing requirements which makes them hard to test. As a result, design problems require a way of thinking that can support dealing with these wicked problems. Cross (2011) explains how designers think based on multiple interviews, research experiments and observations, described next.

The Design Thinking Process

The problem is actively re-formulated. Designers define the problem that needs to be solved, which may be different from the problem given: “*Goals are set at high level, with clear objectives and direct terms... It is this simple clarity which might make other people conclude that the goal is simply impossible*” (Cross, 2011, pp. 73). Perhaps Cross is pointing that designers acknowledge that the current challenge is a wicked problem, and as a result making it high level helps them see the big picture.

There are periods of intense activity followed by reflective contemplation. Cross (2011) discusses that designers will engage in design activities in an almost obsessive fashion, and then completely slow down to reflect on what took place.

Solution strategies are devised abruptly. Cross (2011) states that a strategy to solve the problem “*is achieved by means of sudden insight which comes when relaxing after deep immersion in the problem, and the solution details then cascade from the concept*” (pp. 74). Designers create new patterns that then further re-formulate the problem while suggesting directions for a solution – problem and solution co-evolve.

Methods are non-systematic and done in parallel. Cross (2011) stipulates that design activity continues at many levels simultaneously, in which drawing acts as a primary thinking tool providing different views and different levels of detail. Designers also frequently build models and mock-ups.

Strategic Aspects of Design Thinking

Cross (2011) further describes three strategic aspects of design thinking:

Approaching problems as ‘broad systems’. Cross means that designers think about the bigger picture and how the different pieces relate to each other. He exemplifies this by describing how designers might think of a car in terms of the different parts that make it go, while engineers might focus on designing and perfecting a specific part, such as a clutch. This view is important, as it shows that designers are thinking about the big picture, but also the relationships between the different elements.

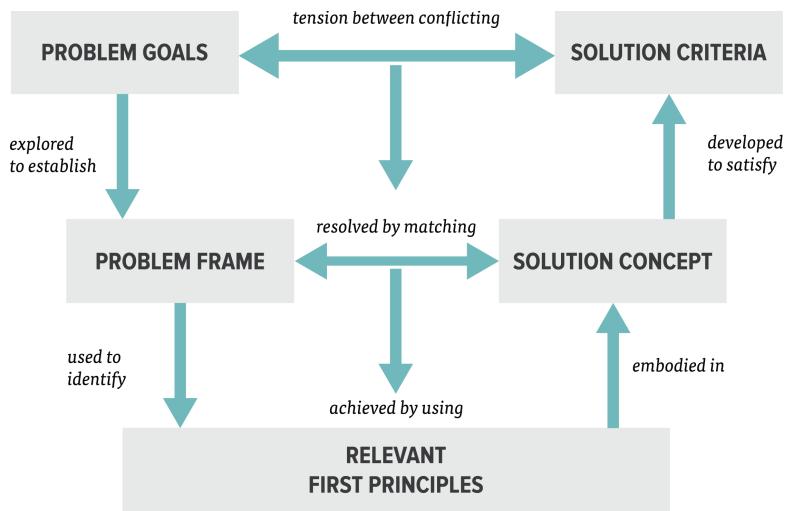


Figure 2.3 Design strategies of creative designers as described by Cross (2011). Diagram adapted from Cross (2011, pp.78)

Framing the problem in a personal way. Cross discusses how designers will frame problems based on the requirements of a situation, but the solution will be strongly influenced by their personal motivation (e.g. altruistic vision of pleasing potential end-users).

Designing from First Principles. Cross explains that designers constantly identify and inform their design through first principles. In Human–Computer Interaction literature, there is a similar concept, known as design guidelines, which are descriptions that “*provide directions for designers or highlight factors that should be considered when designing interactive systems*” (Wiberg and Stolterman, 2014, pp. 533). Wiberg and Stolterman (*ibid*) add that the goal of design guidelines is to specify and formulate factors that a design needs to consider. Thus, designers come with their existing knowledge, but also continue to learn by doing – essentially *experiential learning* (Kolb, 1984), and further inform decisions from other sources, including general research, user research, evaluation results, etc. Cross (2011) adds

that designers use first principles implicit or explicitly in their designs. Indeed, given that designers frame problems in personal ways and each have different experiences, they might be implicitly leveraging these views as first principles throughout the process. Alternatively, designers might learn from different sources, such as user research, or in their own experimentations what the successes and failures of each solution might be.

These thinking processes and strategies, summarized in **Figure 2.3**, help understand how designers approach situations, and how they fit within the larger design process. The thinking process shows how designers need to somehow explore ideas to let these problems and solutions co-evolve. The next section shows that within the design process, designers have a dedicated phase where they actively take on this exploration.

2.1.6 THE DESIGN PROCESS: HOW INTERACTION DESIGNERS WORK

When discussing the design process, the literature can refer to two different workflows: the product development process as a whole, or the design phase within that process.

The Product Development Process

The larger product development process encompasses different phases such as user research, design, implementation, and evaluation. The product development process varies across organizations

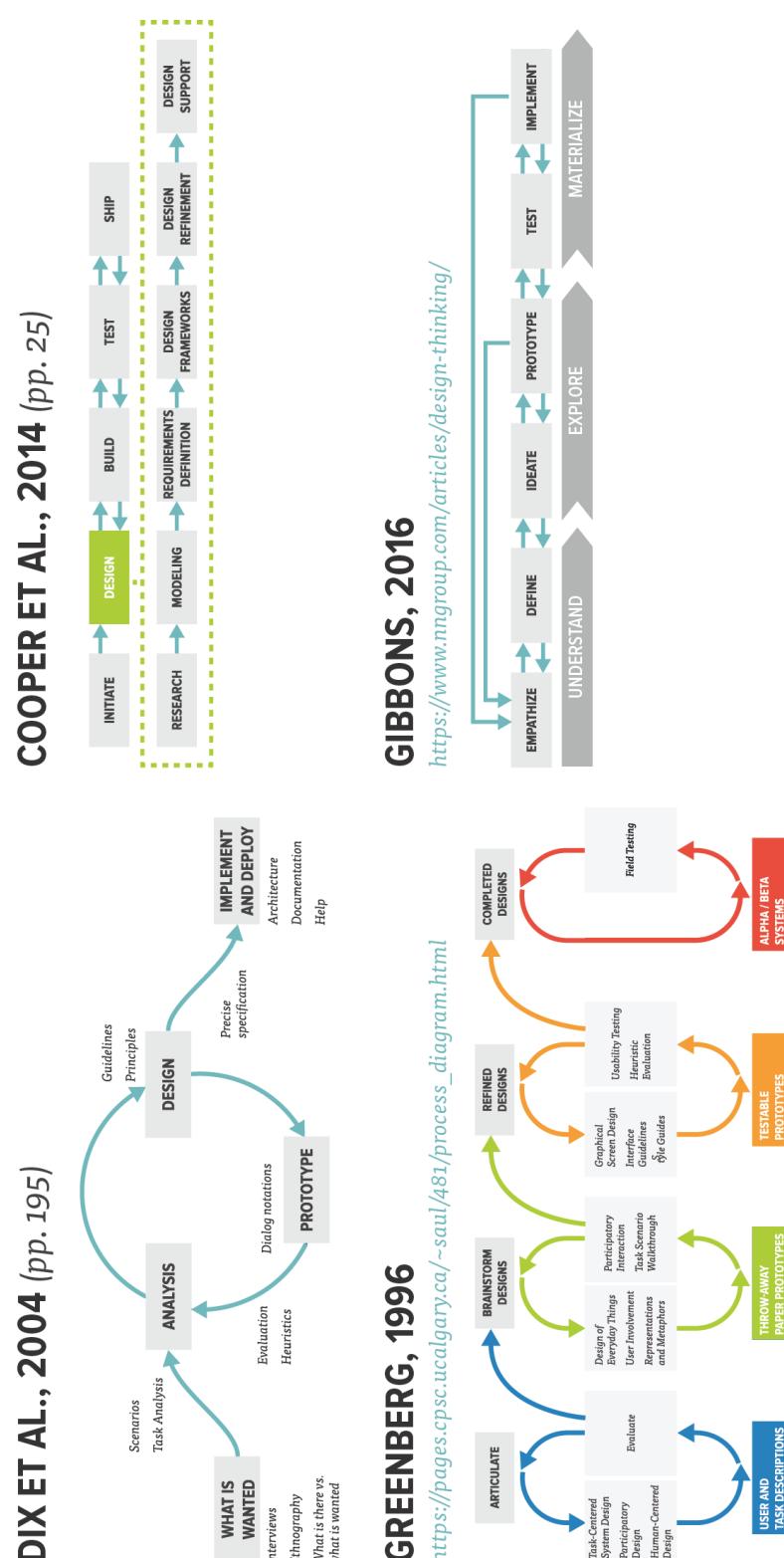


Figure 2.4 Different examples of the design process, as explained by Dix et al. (2004), Cooper et al. (2014), Greenberg (1996) and Gibbons (2016). Diagrams adapted from the respective sources.

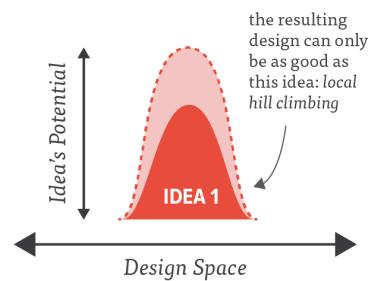
and literature. In fact, Dubberly³, a software design consultancy company has compiled a document with over one hundred different models of the product development process across companies, institutions and academic publications, and it shows how different organizations have different expectations as to what activities an interaction designer might tackle (e.g. user research). *Figure 2.4* shows some different example instantiations of the process, all of which involve designers generating ideas and creating prototypes within a “*design phase*”, the focus of this dissertation.

The Design Phase

The second way in which the literature may refer to the *design phase* itself within the process, which consist of the activities that unfold from the designer generating ideas to arriving at a potential solution under certain constraints (e.g. time, budget, particular idiosyncrasies of the goal). These artifacts are later implemented by someone else (e.g. software developers) and the process is often referred to as *design handoff*. Cross (2011) describes a process followed by experienced designers from an experiment, which included the following steps: (1) quantifying the problem, (2) generating concepts, (3) refining concepts, (4) selecting a concept, (5) designing, and (6) presenting. These steps resonate with other discussions of the design process, including Buxton (2007), Laseau (1982) and Pugh (1991).

Buxton (2007) explains design has two main facets – *getting the right design* to *getting the design right* (explained in *Figure 2.5*). The first step of the process, is to thus generate a multitude of ideas which to

Getting the design right is about starting with a design idea and continually improving it. The best possible version of that idea lies at the hill's peak.



Getting the right design is about generating multiple ideas, selecting between them and refining the fruitful ones.

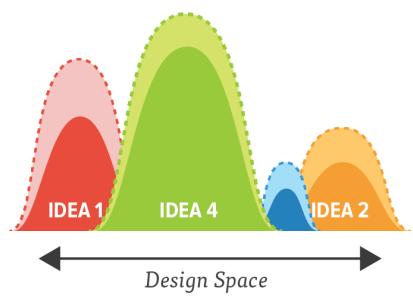


Figure 2.5 *Getting the design right* vs. *getting the right design*, paraphrased from Greenberg et al. (2011)

³ <http://www.dubberly.com/articles/how-do-you-design.html> – accessed March, 2019

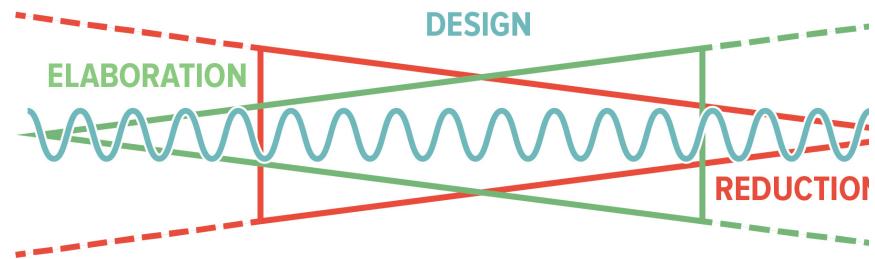


Figure 2.6 Laseau's view (1982) of the design process as elaboration and reduction.
Diagram based on illustration by Greenberg et al. (2011).

find a viable solution. As the process of idea generation moves forward, ideas may evolve in different ways. Rosenman and Gero (1989) suggest that design ideas might result from *combination* (taking features from existing designs), *mutation* (modifying elements of existing designs), *analogy* (making associations outside the current domain), or *first principles* (see Section 2.5). Cross (1997) in empirical studies corroborates these approaches and adds the concept of *emergence* – where ideas evolve by recognizing emergent behaviours in structure or function. What is particularly interesting about emergence is the implication of evolution of ideas, as well as how the four previous elements of combination, mutation, analogy and first principles might come together. The result is that *ideas do not exist in isolation, they inform each other*. Frank Chimero, a professional interaction designer, describes his process in a similar fashion:

“The bad ideas have been documented and captured in some way, which turns them into a resource that can be mined in the process. New and better ideas will certainly come as well, but mixing the two speaks to the cumulative nature of improvising and the special sort of presence it requires. Ideas build on top of one another, and to do so well, one must be in the moment, actively poking at the

current situation to use its opportunities as material for construction.” (Chimero, 2012, pp. 40)

Looking at emergence as a descriptive model of creative design helps understand how the design process might arrive at a potential solution to the problem. Laseau (1980) describes design as a process of elaboration and reduction, diagrammed in **Figure 2.6**. Greenberg et al. (2011) describe Laseau’s process as generating solutions, while also deciding which of those ideas are worth pursuing and further developing those solutions. Pugh (1991) further describes the design process as a funnel, shown in **Figure 2.7**, where cycles of elaboration and reduction converge towards a final solution, until a single concept, which as a result of the convergence has been refined and developed, is selected. Thus, because ideas inform each other, and in the thinking process (see §2.1.5) the problem and solution are co-evolving, the design process naturally evolves towards discovering more fine-grained details of what will become the resulting solution.

In this process, interaction designers create prototypes, which Lim et al. (2008) describe as *manifestations* of ideas, which get developed in

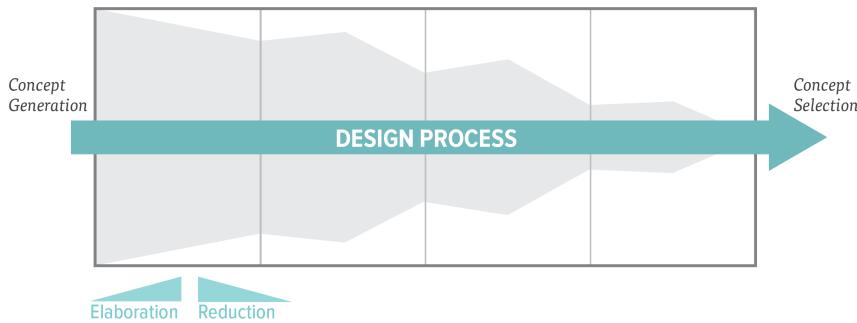


Figure 2.7 Pugh’s description of the design process as a funnel (1990) with multiple cycles of elaboration and reduction that converge towards a solution. Diagram based on illustration by Greenberg et al. (2012).

different *resolutions* (degrees of sophistication) depending on the type of knowledge the designer is trying to obtain from each.

2.1.7 SUMMARY

Design is the structured process for which things are intentionally made under specific constraints. User Experience Design focuses on the quality of the overall experience. According to Cooper et al. (2014), User Experience Design is constrained to digital artifacts, where it is the result of the intersection of form, content and interactive behaviour – the latter which is the focus of interaction designers.

Interaction designers have a wide variety of different backgrounds given its multi-disciplinary nature. However, only few interaction designers are specialized in fields of technology (e.g. information technology or computer science). Given that design problems are often “*wicked problems*”, designers are trained with unique ways of thinking and strategies. Designers follow a non-systematic approach, where they generate multiple ideas, and where the definition of the problem and solution co-evolve.

The product development process often has a dedicated *design phase* where interaction designers explore and devise solutions to problems: they explore multiple ideas, which might come from different places and inform each other, and are externalized and manifested in different degrees of sophistication. The different explorations inform each other, meaning that they do not exist in isolation, rather the solutions emerge from the different idea manifestations. These manifestations are called prototypes.

2.2 PROTOTYPING: HOW DESIGNERS EXPLORE IDEAS

Different design disciplines have particular ways of exploring, navigating and evolving through ideas. Architects might draft blueprints, put together maquettes, create three-dimensional CAD⁴ models; industrial designers might create visual renderings or make physical models out of foam which are sanded down; and graphic designers may print out early versions of their visual arrangements. All of these are different forms of externalization, where the physical manifestations make it so the “*world can speak back to [the designer]*” (Schön, 1987). When describing interaction designers, Lindell (2014) states that interaction designers have a feel for how a design might be realized, which is “*obtained by transforming design into technology*” (pp. 617). Lindell adds that interaction designers create realistic representations in a quick and at times chaotic manner. These manifestations of ideas are all different kinds of prototypes. I next describe what prototyping is (§2.2.1), why designers prototype (§2.2.2), the aim of prototypes (§2.2.3), and how it is possible to deconstruct those aims as structure, behaviour and usage (§2.2.4).

2.2.1 WHAT IS PROTOTYPING?

To a lay person, prototyping might mean “*a first or preliminary version of a device or vehicle from which other forms are developed*”⁵, but prototyping is much more than a single preliminary version. A more suitable definition for prototyping is Lim et al.’s (2008) take: “*prototypes*

⁴ CAD stands for Computer-Aided Design

⁵ <https://en.oxforddictionaries.com/definition/prototype> – accessed January 2019

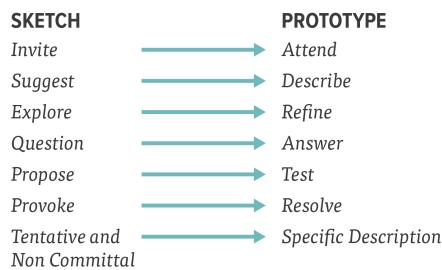


Figure 2.8 Buxton's (2007) distinction between sketching and prototyping. While ideas in design indeed naturally evolve from exploration to specificity, the design literature still largely refers to all of these techniques as prototyping. Figure reproduced from Buxton (2007).

are a tangible attempt to view a design's future impact so that we can predict and evaluate certain effects before we unleash it on the world" (pp. 8).

Beyond Schön's view that design is a reflective practice – a conversation with the situation (1987), Lindell (2014) argues that design, through its externalization, is a form of craftsmanship: "*patient and not tempted to do quick fixes*" (pp. 617). Lindell further cites Sennet's view of crafting (2009), where problems are identified in the making process and solutions are identified simultaneously: a relationship between hand and mind.

2.2.2 WHY DO DESIGNERS PROTOTYPE?

Because prototypes are different forms of externalizations, they can serve different roles in the design process. The next subsections discuss how prototypes can be used for exploration, specification and communication, as well as evaluation.

Prototyping as Exploration

Because Human-Computer Interaction and Interaction Design draw from multiple disciplines, including design, software development, and engineering, prototyping has different interpretations. To remedy this, Buxton (2007) distinguishes between prototypes and sketches to illustrate different roles, as shown in **Figure 2.8**. Buxton creates a distinction where sketches are exploratory and prototypes are meant to specify. Yet, much of the design literature, including Goel and Pirolli (1992), Logan and Smithers (1992), Lim et al. (2008) and Cross (2011), advocate for prototyping as a means to design thinking in the same way that Buxton discusses sketching: prototypes

are “*tools for traversing a design space where all possible design alternatives and their rationale can be explored*” (Lim et al., 2008, pp. 2) as opposed to means to “*identify and satisfy requirements*” (*ibid*, pp. 2).

Logan and Smithers (1992) warn against seeing prototypes as mere of parametric descriptions, or seeing them as means to generate descriptions, as this view can lead to two flawed assumptions: (1) prototypes have little or no link between each other, and (2) solutions can be achieved through search. Instead, it is not through the prototypes themselves, but through the exploration activity itself that gives designers such an understanding, as it suggests the breadth of possible solutions, and conveys whether the solution might be feasible given the current constraints. Lim et al. (2008) add that the strength of a prototype lies in the fact that it is incomplete: “*it is the incompleteness that makes it possible to examine an idea’s qualities without being a copy of the final design... [it] structures the designer’s traversal of the design space by allowing decisions along certain dimensions*” (pp. 7).

The reason why prototypes shift towards something that appears more like a specification can be explained by the design funnel of §2.1.6 – ideas evolve and inform each other, which leads to more set decisions. The types of manifestation might shift over-time, and likely take on *higher resolutions*. However, level of sophistication may not always correlate to the stage of the design process (Houde and Hill, 1997).

Prototyping as Specification and Communication

Prototyping can be seen as a means of communication between designers, developers and stakeholders (Sharp et al., 2015). Floyd

(1984) explains that unlike traditional manufacturing, software systems have often unspecified and often changing requirements, and prototyping is a means of discussion to explore and define the requirements. Moreover, prototyping can support rapid feedback cycles between designers and clients, communicate new ideas to developers and answer unanswered questions (Gerber and Carroll, 2012). Thus, the prototypes become a shared instrument that designers can leverage to devise solutions (Bødker and Grønbæk, 1991). Indeed, Bertelsen (2000), as well as Leiva (2018), describe prototypes as *boundary objects* (Star and Griesemer, 1989) between communities of practice, as designers can ground conversation between different groups of people (developers, clients, users, etc.). As the design cycle progresses, the communication might also become more a form of specification, where designers can clarify their intent with developers. In fact, studies of the communication between designers and developers show that designers will go as far as resorting to high-end video editing tools such as Adobe AfterEffects, to describe nuanced animations, transitions and interactions to developers which are then supplemented with explanations (Maudet et al., 2017).

Prototyping as Means of Evaluation

Prototyping is also described as a means of evaluation. By creating early versions of the system (e.g. using paper), it is possible to conduct usability inspection and evaluation methods to more rapidly iterate between versions of the system (Nielsen, 1993). Lim et al. (2008) argue that this view, while more traditional in earlier Human-Computer Interaction approaches, tends to favour standardized graphical user interfaces, and is only one narrow way of looking at

prototyping. Furthermore, the usability inspection of a paper prototype may not be as viable in a post-WIMP (Windows, Icons, Menus and Pointers) approach given that state transitions and the like are more difficult to convey, thus requiring higher resolution solutions.

What these different views on prototyping show thus far is that prototyping is primarily a means of exploration through making, where designers can discover an answer to a question. As questions get answered, the solution takes shape which can additionally be used for communication as well as evaluations. Such questions become more concrete as a result of the cumulative knowledge from the prior prototyping activity.

2.2.3 WHAT DO PROTOTYPES PROTOTYPE?

While the last section describes the different uses and benefits of prototyping, this section explains shows how prototypes answer specific questions. Indeed, Lim et al. (2008) describe prototypes as filters – a way to view the problem and explore the answer to a question.

Given multiple sources from a variety of disciplines (e.g. engineering, computer science), prototyping has different interpretations as to what questions they might answer. However, many of these discuss elements outside of design (e.g. technical implementation).

Gero (1990) described prototypes as parametric descriptions that define *function*, *structure*, *expected behaviour*, and *actual behaviour* of an object. Licher et al. (1994), on the other hand, described prototyping from the point of view of software development: a *presentation prototype* is used to convince potential clients that a problem can be solved; a *prototype proper* shows functional elements to clarify the problem at

hand; *breadboards* examine implementation details to help software specification; while a *pilot system* illustrates how the software works and enables early experimental testing. Houde and Hill (1997) posit prototyping in terms of what they might be trying to articulate when creating software applications, a prototype might prototype: *implementation*, examining how to solve a software problem; *role*, investigating how a system might be used in the real world; and *look and feel*, referring to the visual and behavioural elements within the system. Houde and Hill argue that these prototyping angles are not mutually exclusive, and that a prototype might investigate the answers to these questions in different levels. Houde and Hill also distinguish between *fidelity* (stage within the design process) versus *resolution* (level of sophistication of the prototype), as it is not always the case that the level of sophistication only increases as the design process moves forward.

Lim, Stolterman and Tenenberg (2008) propose a similar and more nuanced view of prototyping to Houde and Hill, though much more focused on broader aspects design (e.g. industrial design), especially designing physical form. Lim et al. provide a framework for thinking about prototyping in terms of an *anatomy* composed of *filters* and *manifestations*. Prototypes as **filters** focuses on what aspect in particular the prototype is trying to explore, which can include: *appearance* (physical properties such as size, colour, shape, etc.), *data* (information architecture, such as the number of labels, content organization), *functionality* (system functions and user needs), *interactivity* (behaviours in terms of input, output, feedback, and information),

and *spatial structure* (how components are combined, such as the spatial arrangement of the interface). As **manifestations** (i.e., as artifacts), prototypes might have variable *materials* (medium), *resolution* (level of sophistication), *scope* (range of what is covered).

2.2.4 EXPLORATORY PROTOTYPING IN INTERACTION DESIGN

Because the views on what prototypes might target are so broad, I am taking the relevant points of what prototypes aim to achieve when it comes to exploration in interaction design. Given interaction designers' training (see §2.1.4), they are not concerned with the specific details of how a system or artifact might eventually be implemented (i.e. examining functionality and actual implementation details). The key takeaway is that this subsection contextualizes the focus of this thesis, namely the exploration of interactive behaviour (defined in Chapter 3), as we need to better understand: what do interaction designers need to consider to explore ideas in interaction design? Note that this is not an exhaustive categorization. All these elements are interconnected in some way – a prototype's structure defines the possible behaviours, and possibilities of trying it out. As a result, to test the behaviours, there needs to be some form of underlying structure (e.g. an interface mock-up) of some resolution or fidelity from which the behaviour can be the basis. This is summarized in **Figure 2.9**. Given that this outline of exploratory behaviour builds on what prototypes prototype in different areas of design, computer science and engineering, **Figure 2.10** shows how all of these elements are being integrated. Note that in particular, implementation details are not in scope of idea exploration.

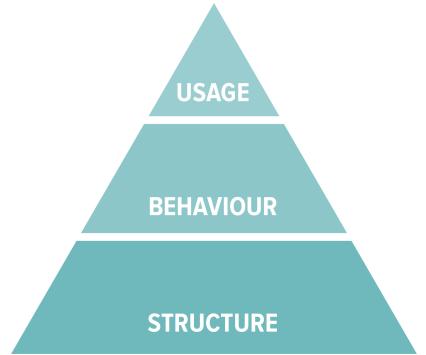


Figure 2.9 Schematic of this thesis' exploratory prototyping in the context of interaction design. Given existing research on what prototypes prototype, I argue that exploratory prototyping has the components of Structure, Behaviour and Usage. Structure is the basis (form or layout) which the person interacts with. Behaviour is what the system does before, during or after the interaction. Usage refers to a person's ability to try out the behaviours given the provided structure.

		EXPLORATORY PROTOTYPING IN INTERACTION DESIGN		
		STRUCTURAL ELEMENTS	BEHAVIOURAL ELEMENTS	USAGE ELEMENTS
GERO (1990)	Function			
	Structure	■		
	Expected Behaviour		■	
	Actual Behaviour		■	
LICHTER ET AL. (1995)	Presentation Prototype	■	■	■
	Prototype Proper			
	Breadboard			
	Pilot System	■	■	■
HOUDÉ & HILL (1997)	Implementation			
	Look and Feel	■	■	
	Role			■
LIM ET AL. (2008)	Appearance	■		
	Data	■		
	Functionality			
	Interactivity		■	
	Spatial Structure	■		

Figure 2.10 Exploratory prototyping in the context of prior work which discusses what prototypes might prototype. Note that in the context of interaction design, reflecting system-level implementation is not in the scope of the prototyping practice.

Structure: The What

Physical Form. The most basic forms to work with in interaction design would be to assume one is working with existing types of devices, such as a desktop or laptop computers, or some type of mobile device. Forms can become more complex, especially when considering an appliance such as a radio. The form itself also provides users with the structure of the input and output. For example, a radio may have knobs and sliders that the user can manipulate as inputs, and the speakers may change volume and react accordingly. The placement and arrangement of the controls matters as well.

Visuals and Layouts. When working with digital screens, users are dealing with a digital counterpart to the form. The structure is defined by the layout of the interface and the types of controls provided to the user. What makes the visual and layout more complex is that the contents are dynamic – they can change any time.

It is also worth mentioning that the structural elements of form and layout are very similar, though form is more related to physical arrangement of controls, while layout is more related to the digital arrangement of the different controls. An artifact may feature both structural elements, such as a radio with a digital display.

Behaviour: The How

The structural elements outline “*what* people interact with”, whereas behavioural elements look at “*how* people interact with the *what*”. Thus, interactive behaviour is ultimately about how the inputs become outputs, all which tie back to the structure of the device/artifact/software. Interactive behaviour will be explained further in depth in Chapter 3.

Usage: First-Hand Experience with the Concept

To get a sense of how the behaviours work, the designer needs to envision them in action, and perhaps even try them out. A prototype might also be created to explore how it might be used. The evaluative approach to prototyping looks at giving a prototype to users that they can test, but from an exploratory perspective, *designers can execute ideas that they can try out and foster self-reflection to arrive at new first principles*. The most basic form of usage requires little behavioural elements, as it is fostered by imagination. For example, Jeff Hawkins (Moggridge, 2007) carried a block of wood as a stand-in for the Palm Pilot, and would pretend to use it throughout the day to understand how it might work (e.g. for scheduling, setting reminders during meetings). However, as one needs to understand more complex behaviours that are less tied to usability, such as how a system provides feedback, or other experiential elements such as what happens at a system-level as a result of continuous actions (e.g. how a display might show feedback of interaction with a slider), it is necessary to have a more fleshed out interactive behaviour within the prototype.

2.2.5 SUMMARY

In the design process, designers create prototypes as a way to manifest their ideas and explore an individual question. These manifestations take different forms, are made in different resolutions (degrees of sophistication) and at different fidelities (how early/late the design process). While prototyping theory has been explored in many different areas aside from design, including engineering and computer science, prior work presents different ways in which prototyping can be

used, including exploration, communication/specification and evaluation. Within these types of prototyping, the activity itself may answer different questions such as the appearance, how it works, etc. I argue that for exploratory prototyping in interaction design, prototypes might answer questions of structure (i.e. the visual layout and the physical form), behaviour (how inputs become outputs) and usage (ability to get first-hand experience with the concept). In the context of Human–Computer Interaction, there is a better understanding of prototyping form and visual layouts, and how to inspect a system's usability. However, the concept of interactive behaviour still remains vague, which is further explored in the next chapter.

CHAPTER 3. A DESCRIPTIVE FRAMEWORK OF INTERACTIVE BEHAVIOUR

Indeed, to *propose methods for interaction designers to prototype interactive behaviours for smart interactive objects* entails an understanding of what is meant by interactive behaviour. The concept of interactive behaviour, while on the surface may seem simple, is actually quite a complex conversation – perhaps one that merits extensive exploration, as it may be no different to bigger discussions in the community such as the meaning of *interaction* (Hornbæk and Oulasvirta, 2017) or the meaning of *interactivity* (Janlert and Stolterman, 2017). Look and feel of an interface are often grouped together as if they were a single unit. Myers et al. (2008), however, realized that not everything was about the layout of a user interface in software, which prompted further investigation on the subject. The solution was to separate “*look*” from “*feel*” to get to a closer grasp of interactive behaviour. Still, to prototype interactive behaviours, it is necessary to define what interactive behaviours are.

The previous chapter introduced interactive behaviour as a part of exploratory prototyping in interaction design and how it is dependent of the underlying structure of a system or artifact (i.e. the physical form or visual layout) and enables people to have first-hand experience with the system or artifact. In this chapter, I bring together different theories of HCI to help define interactive behaviour and understand how researchers might describe these kinds of behaviours. Understanding what behaviour means is a fundamental step in HCI research if we are to design the next generation of tools to generate interactivity beyond code. It is especially important if '*behaviour*' continues to be broadly used, and oversimplified in research discussions. To address these needs, I explain why interactive behaviours are not simply synonymous with the "*feel*" of an interface (§3.1), define interactive behaviour (§3.2) and navigate through different fundamental theories that explain how inputs and outputs come together to create dynamic and responsive experiences (§3.3), which provide insight on the small nuances when designing these types of behaviour.

3.1 THE PROBLEM WITH THE WORD FEEL

Interactive behaviour is often described as "feel", which leads to many loose interpretations and challenges within the field – it is not a term that can be easily operated on. Indeed, the elements of interactivity are often lumped together with visual elements when discussing prototypes in terms of "look and feel", such as Houde and Hill's view on prototyping (1997). However, this is not in discredit to them, as at the time desktop interfaces had reached a high degree of standardization given the prevalence of the WIMP (Windows, Icons, Menus and Pointers) paradigm and different user interface widgets

(e.g. evolved versions to the ones in the Xerox Star (Johnson et al., 1989)). Myers et al. (2008), investigated how designers author behaviours and tried separating “look” from “feel”, by defining *feel* as “... anything that an application does... what you cannot draw... anything that required [authoring] using [a] timeline or scripting” (pp. 1). The word “feel” is easily open to misinterpretation because it can refer to many non-interactive elements associated to it: a system might feel smooth, a system might feel familiar, a system might feel unresponsive, a system might feel dated, a system might feel modern or old, etc. and many of these feelings can be evoked exclusively via aesthetic choices. For example, the visual style choices in video games such as using pixelated artworks can make a game look and feel “retro”, while an artifact might also have tactile qualities such as material and texture (e.g. feeling soft, fuzzy, plastic or metallic), which again refer to a different kind of feel. Indeed, these aspects are important, and will affect the overall user experience, but are distracting from what prior work has tried to describe if the description lacks precision: designing interactions with a system or artifact beyond the structural elements of physical form and visual layout.

3.2 INTERACTIVE BEHAVIOUR, DEFINED

Given a system or artifact’s layout, the interactive behaviour is what allows a person to actively engage and interact with the system or artifact. Thus, from a design perspective, interactive behaviour can be defined as follows:

Interactive behaviour is how a designer defines a series of inputs to become a series of outputs.

When people interact with a system, they perform (implicit or explicit) *actions* which are interpreted as one or more *inputs* (captured by *sensors* such as buttons, microphone, camera, accelerometers, touch screens; *input devices* such as mice or keyboards; or *contextual elements* such as time). The system responds to these inputs via *outputs*, which may be conveyed via visual displays, sound, tactile and haptic feedback, etc.

Having this definition helps us narrow the scope into a definition that is operationalizable by designers. Moreover, it takes the focus away from the end-user in terms of how they might perform the actions or how they might interpret them, and makes them fully about what a system or artifact can do with its ability to receive information from the world (inputs), interpret it (via computation or mappings), and respond (output).

For example, consider a light switch which is attached to a ceiling light. From a people-centric perspective, a person performs the action of flipping the switch which in turn makes the lights go on. From a design perspective (and under the current definition) the switch has the ability to sense two values, which are interpreted and mapped into two possible values (on or off), and the response involves switching the current state. If the light switch is more complex, such as having a dimmer, or a colour slider, the interpretation and mapping also increases in complexity, leading to a wider variety of possible, and perhaps dynamic, responses.

Moreover, this definition of interactive behaviour fits Beaudouin-Lafon's (2004) evaluation metrics for interaction models, as it as:

Descriptive Power. It can describe and fit within existing interaction paradigms from interaction with desktop computers to interaction with smart objects. The light switch example shows how indeed, the definition can describe many variants of interactivity, a variety of inputs (e.g. a simple switch versus sliders or even a mobile application), and outputs (e.g. on/off state, brightness and colour).

Evaluative Power. Designers can use this definition to assess interactive systems, and they can confirm or deny whether their intention was met. If the designer defines a slider value to dim lights, they can assess via first-hand experience whether the lights change brightness as the action is performed.

Generative Power. This definition of interactive behaviour can be used by designers to generate new designs. In fact, the systems and concepts built with this thesis from Chapter 6 onwards are examples of how this definition can generate new designs.

3.3 A DESCRIPTIVE FRAMEWORK OF INTERACTIVE BEHAVIOUR

To create a framework means to provide a conceptual contribution in HCI. Rogers (2004) describes the role of theory in HCI as providing different types of knowledge. Rogers (*ibid*) states that theories can be: (1) *informative* (provides useful research findings), (2) *predictive* (can model user behaviour), (3) *prescriptive* (provides advice for design or evaluation), (4) *descriptive* (provides rich descriptions), *analytic* (identifies problems), (5) *formative* (provides concepts to discuss designs), and (6) *generative* (provides constructs that can foster a variety of solutions). I bring together different components of existing HCI

theories to describe interactive behaviour, which provides interaction designers and other HCI researchers with a unified vocabulary. In particular, this framework contributes:

1. ***Descriptive elements***, as it outlines many components to interactive behaviour,
2. ***Formative elements***, as it brings terms and vocabulary used in foundational HCI theories of input, and to a lesser degree,
3. ***Generative elements***, as designers could use the vocabulary provided and generate different kinds of ideas for interactive behaviours

While the notion of *interactive behaviour* is not explicitly talked about in Human–Computer Interaction, there are fundamental theories examining models of interaction, as well as models of input, which can help in defining what interactive behaviour means. Current work shows that behaviours (1) are a relationship between inputs and outputs, (2) they have dependencies from those inputs and outputs as well as a state, and (3) they are heavily influenced by programming paradigms given that they are always the result of some degree of programming.

3.3.1 BEHAVIOURS ARE RELATIONSHIPS

BETWEEN INPUTS AND OUTPUTS

In Human–Computer Interaction, behaviour is typically described as a relationship between the user and one or more objects on screen (or an object of interest beyond the screen). Thus far, the considerations of behaviour discussed have focused primarily on the input, without

making much mention to the contents of the screen. In Human–Computer Interaction, the concept of *direct manipulation*, as defined by Shneiderman (1983), became a way to understand that computers could be more than a console one types into – they were capable of rendering objects one could directly operate upon. The direct manipulation paradigm looks to have continuous representation of objects of interest, to which one can apply “physical” actions (e.g. clicking and dragging), allowing rapid and reversible operations that could be reflected on these objects of interest. Maloney and Smith (1995) argued for interfaces to be live and direct, active and reactive, in what they called morphic user interfaces. Morphic interfaces were supported under four implementation techniques: structural reification¹, layout reification, ubiquitous animation, and live editing. Many of the standard interactions afforded by devices and operating systems today follow this direct manipulation paradigm. Instrumental interaction, as described by Beaudouin-Lafon (2000) provides a means to incorporate the different elements present on the screen which are defined as *domain objects* (potential objects of interest). Instrumental interaction considers *interaction instruments* (e.g. a scrollbar or any UI widget) as mediators in the process of converting a *user action* (moving the mouse to the scrollbar) into a *command* (scroll) that can appropriately affect the *object of interest* (a textbox). Note that this framework was built under the assumption of direct manipulation, and some of its articulation becomes more difficult to interpret as one

¹ Reification refers to making abstract things more concrete. Oxford Dictionary.
<https://en.oxforddictionaries.com/definition/reify> February, 2019

shifts to other platforms such as smart objects. Still, the notion of object of interests as being entities beyond the screen remains – perhaps in the form of an LED, a sound, etc.

3.3.2 BEHAVIOURS HAVE DEPENDENCIES

Pioneering work in Human-Computer Interaction examined different types of input devices. While the primary context was desktop computing, it also spanned different kinds of everyday devices and appliances. The 1980s and 90s show different taxonomies aimed at understanding input devices and from that understanding be able to derive interaction techniques (defined by Foley and van Dam (1990) as “*a way of using a physical input/output device to perform a generic task in a human-computer dialogue*”).

Which behaviour is active depends on system-level global and local states. Buxton (1990) proposed a three-state model of graphical input (Figure 3.1), which describes the actions of a cursor based on what the input device does as a means to (1) appropriately match devices to input techniques, and (2) compare different devices or techniques to each other. As such, a mouse has the states “*tracking*” and “*dragging*” (depending on whether the button is being pressed). A pen has an additional state which is when the stylus is “*out of range*”, and touch events are either “*out of range*” or “*dragging*”. Thus, different *quasimodes* (or user-maintained modes (Raskin, 2000)) are transacted depending on the current selection. Buxton (1983) lists: pointing, tracking, selecting, dragging, rubber banding, menu pulling, character recognition, and inking as examples. Note that this model described the interfaces at the time and takes the cursor-based interac-

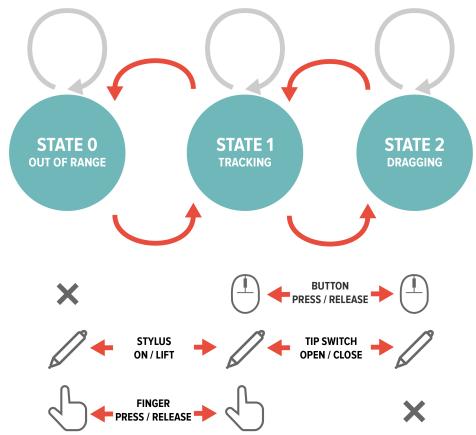


Figure 3.1 Buxton’s (1990) three-state model of graphical input showing the three states: (0) *out of range*, (1) *tracking*, and (2) *dragging*. It also shows example inputs of mouse, stylus/pen, and touch and how they might transition between each state. Figure based on Buxton (1990).

tion as the main assumption while categorizing different kinds of devices. It does not discuss what kind of sensing they provide or how that sensing is mapped to a particular output.

Traditional computing systems, as well as systems that rely on preserving some degree of familiarity of such desktop-like systems, rely on global states. For example, a drawing tool, even on a multi-touch tablet device, typically has a global mode it ties to, which defines the current drawing tool. As new inputs become available, it is possible to look at input in a local manner. For example, in Local Tools (Bederson et al., 1996) and Constructible Interaction (Walny, 2016), tools are objects with can be dragged and acted upon individually. Walny (2016) demonstrates that through visibility and locality it is possible to have multiple objects which perform specialized operations and could work across different collaborative settings. Once the input moves away from the single cursor, global modes can be somewhat relaxed in favour of states that affect only what the user is interacting with. Collaborative tools, where multiple people are entering inputs, or tools that, for instance, leverage bimanual interaction or multiple inputs, operate best with local states. Localizing states makes it so that an input can be deconstructed and mapped to modify specific parameters of the outputs, as done in systems such as ICon (Dragicevic & Fekete, 2004). The reason for this is that the task is no longer tied to a single point of interaction which forbids the existence of others. That said, there can be a global state or mode (e.g. whether one can move objects on a screen versus only view them) which guide the actions of all inputs and outputs.

Inputs and outputs have properties/parameters. Mackinlay, Card and Robertson (1990) brought another taxonomy that captured discrete and continuous properties sensed within input devices (as informed by existing toolkits to date): devices can sense position or force in an *absolute* or *relative* fashion, and inputs can be linear or rotary. Mackinlay et al. represent input devices as a six-tuple of: manipulation operator, input domain, state of the device, resolution function, output domain and specific device properties. The taxonomy describes devices as both physical (e.g. mouse) or virtual (e.g. cursor). The output domain set of one device can be composed into the input domain set of another, referred to as a *connection*. An example of this connection is a radio in which the rotation of a station knob affects a physical slider showing the current station. **Figure 3.2** shows an example by Mackinlay et al. (1990) of a radio and how the inputs can affect each other. In particular to this example, the selection of AM and FM, and the station slider value are combined into a single value that represents the current station.

Inputs are mapped into outputs. Hinckley et al. (2014) put together some of these ideas in terms of how input devices behave, in particular some of their added relevant properties pertaining to how the inputs can turn into outputs include:

- **Property sensed.** absolute or relative values sensed by the input device (e.g. change in position sensed by a mouse).
- **Transfer function.** the device and operating system apply a mathematical function to the system to scale the data and “provide smooth, efficient, and intuitive operation” (pp. 8). Design-

ers create *appropriate mappings* when matching the physical properties of the device into a potential output (e.g. converting a joystick's sensed force to the velocity of a cursor's movement).

- **Number of dimensions.** devices measure linear and angular values as determined by the sensors.
- **State.** Providing the meaning of what the system should do when provided with a new value (e.g. a point of a cursor).

Input and output parameters can be combined and abstracted, implicit or explicit. Another framework that helps contextualize interactive behaviour is Implicit Interaction (Ju and Leifer, 2008). Implicit interaction posits that systems can have a varying degree of *initiative* (reactive or proactive), and that the actions from a system can require different levels of *attentional demand* (take place in the foreground or the background). Other frameworks continued to build on the idea of implicitness, such as Proxemic Interaction (Ballendat et al., 2010). An important element within the work in proxemics is the idea of different spatial relationships (identity, distance, orientation, movement) can dynamically affect the contents of different devices and their interfaces within an artifact ecology. Considering newer forms of interaction, including touch, mobile interactions, etc. leads to the realization that interaction with systems is no longer tied to a single input at a time with perhaps a few modifiers (e.g. mouse and keyboard) but that multiple inputs are affecting the contents on a device regularly (e.g. multitouch, added sensors on a phone), which makes the case for interactive behaviours considering different sources of input, as well as the relationships between those inputs. For example, in the one-handed mobile device interaction technique “*tilt-to-zoom*”

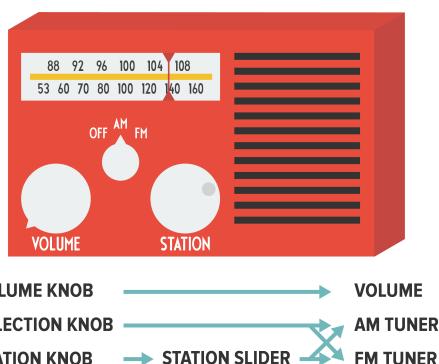


Figure 3.2 Mackinlay et al. (1990) taxonomy as exemplified on a radio. Image based on Buxton (2013).

(Song et al., 2011), one can simultaneously tilt back-and-forth to zoom in or out, as well as pan the finger across the screen to navigate a map in different cardinal directions. Visuals respond to these two dimensions independently.

3.3.3 PROGRAMMING CONSTRUCTS OF BEHAVIOUR

Mackinlay et al. (1990) discussed the influence of their current toolkits in their taxonomy. This is no surprise given that the interactive systems people use are shaped by the language used (exemplified in the next Chapter). Object Oriented Programming and the emergence of user interface builders popularized the idea of events, which provide information when the system detects a specified change (e.g. updated values within an input device's parameters, a key press, cursor entering a selection, etc.). High level tools, such as Expression Blend, have adopted this paradigm, and is referred to as “*triggers*”.

Triggers can act as a way to modify the current *state*.

Time is an input, but not always the most appropriate abstraction. All interactions always take place over time, and there is no question to the importance of time as an abstraction in many areas such as animation. Time might be an appropriate abstraction when animating a reaction (e.g. a screen transition when pressing a button). However, time as an abstraction becomes less useful when displaying information from a particular input (e.g. mouse moving). The reason for this is that there might be a large number of time units (e.g. ticks or steps) in which values do not change, thus, a common abstraction is to talk about isolated events pertaining to the particular object (e.g. mouse move events, button clicked). The consequence has been to

favour *triggers*. Indeed, triggers are appropriate to “schedule” reactions which are animated as time – Hartmann (2009) calls these responses “*one-shot animations*” to indicate how they take place only when the trigger takes place.

Many programming approaches will dynamically map input values to visual elements. For example, when applying a “*pinch to zoom*”, a common interaction in touch surfaces, the distance between touch points dynamically adjusts the size of the object of interest. Hartmann (2009) refers to these as “*user-in-the-loop*” behaviours, as “*continuous user input drives the behaviour*” (pp. 31). Yet these types of mappings, or animations as a function of input parameters, are not yet applied in higher-level prototyping tools. Another type of behaviour which is not talked about and is out of the scope of this thesis is *cumulative behaviours*, such as when drawing on a canvas with a brush tool. In the case of the brush tool, the points drawn are accumulations of a series of inputs which belong to the same cognitive operation, which is known as chunking (Buxton, 1986).

3.3.4 SUMMARY

I defined interactive behaviour as *how a designer defines a series of inputs to become a series of outputs*. This definition helps scope the views on behaviour, as it shifts the focus to how the system’s responds to sensors, input devices or contextual elements (e.g. time) to act in a proactive or reactive manner. I then integrated paradigms from Human–Computer Interaction to outline that (1) behaviours are a relationship between inputs and outputs; (2) they have dependencies

from those inputs and outputs, global and local states, and parameters; and (3) they are heavily influenced by programming paradigms given that they are always the result of some degree of programming.

Overall, the foundations of interactive behaviour lead to particular considerations that need to be satisfied to facilitate its prototyping:

1. Understanding that there is a state that influences the active behaviour, which is optional to a system
2. Thinking of input and output as abstractions with parameters that can be used for mapping (e.g. a slider's value, a light's brightness)
3. Thinking of outputs as animated transitions (pressing a button and having the screen show an animation), cumulative operations (drawing with a brush tool on a desktop application), as well as interaction-driven animations (moving a slider and seeing a colour change dynamically as the mouse is being moved)

3.4 CONCLUSIONS

Overall, this chapter provides a key working definition for interactive behaviour which is founded on the way a system interprets a user action. By looking at different theories and taxonomies of inputs, it was possible to devise additional characteristics of interactive behaviour which shape and describe most interactive systems today. The next natural question is then how to prototype interactive behaviour, which points to the tools and techniques designers use today, as well as the existing gaps in this knowledge.

CHAPTER 4. PROTOTYPING INTERACTIVE BEHAVIOUR, RELATED WORK

Now that the definition of interactive behaviour has been scoped, this chapter describes how designers create prototypes in interaction design. First, I outline some common prototyping activities to provide a sense of existing practices, where I highlight their common resolution and fidelity, as well as the extent to which they achieve prototyping structure, behaviour and usage (§4.1). Moreover, I then elaborate on the challenges of interaction designers, and how the most common commercial tools do not address this challenge (§4.2). I argue that the current commercial tools are limiting the types of prototypes designers can achieve given (1) the focus on static structure and negligence of prototyping interactive behaviour, and (2) the tendency to oversimplify the notions of behaviour. I conclude this section with a review of existing software tools in industry and academia beyond the common tools, and providing an overall taxonomy (§4.3) outlining different authoring paradigms that can help inspire the next generation of tools for prototyping interactive behaviour.

4.1 COMMON PROTOTYPING ACTIVITIES

Interaction designers follow a variety of prototyping activities which lead to different types of discoveries. The next subsections outline examples of different design activities and the type of knowledge that they might yield. As discussed by Houde and Hill (1997) and Lim et al. (2008), these prototyping approaches have varying *resolutions* (degrees of sophistication) and can take place in different stages of the design process, not necessarily tied to a specific *fidelity* (how early or late in the design process). However, what makes these activities different is that they (1) prompt and yield different kinds of questions and answers; (2) require varying degrees of effort and time commitment. Thus, designers use their judgement to decide what is an appropriate manifestation at the time. Typically, there is the association that fidelity and resolution are directly proportional. For example, designers sketch ideas on paper at an early stage, then progress to higher levels of sophistications. However, this is not always the case. For instance, a late stage of design may require ideas on paper again if a change needs to happen, or a designer may jump right away into an expert tool and produce a high-resolution solution that explores one small aspect of the user experience.

4.1.1 SKETCHING

Sketching is an activity that is common in all areas of design, often described as the core activity. By creating quick drawings, designers can rapidly (i.e. in a matter of minutes) generate new ideas and have them documented. Thus, it is a technique where a designer can manifest and externalize an idea in a low resolution. Sketches may have

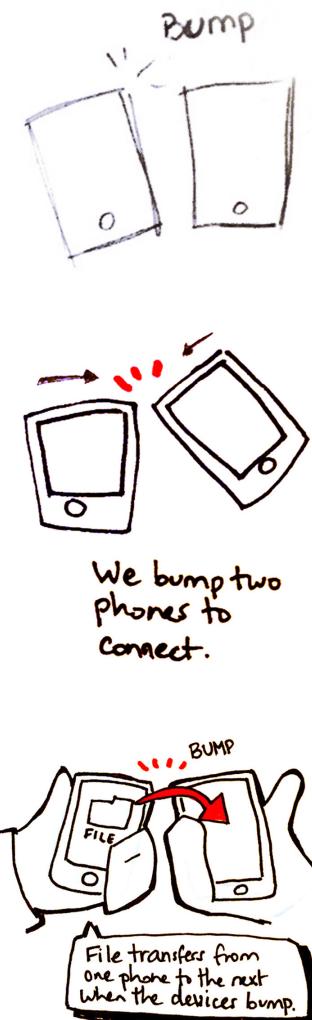


Figure 4.1 Example of quick sketches for a single idea: “how might two phones share a file by using their built-in sensors?” In this case, the sketch shows bumping two devices together to share files. While each sketch shows the same idea, they have different levels of visual detail, none of which took more than two minutes to complete.

varying degrees of depth, which can be expressed by the level of visual detail, or through additional descriptions (e.g. secondary screens, drawings to depict the interaction). Buxton (2007) explains how the resolution of the sketch should depend on how developed the idea is. Baskinger and Bardel (2013) define this as a spectrum from thinking to describing. Stolterman (2008) compiles many authors describing sketches, and concludes that “*Sketching is a disciplined way of exploring the relationships between diverse design ideas, between a whole and details, between form and function, between appearance and materials... a rational designer works on many alternative designs in parallel in an iterative way, while going back and forth between the whole and the details*” (Stolterman, 2008 pp. 61). **Figure 4.1** shows an example of sketches for exploring a design idea.

4.1.2 WIREFRAMES / STORYBOARDS

Wireframing refers to visual representations of the different screens or states of an interactive system. These often specify the different interface elements (e.g. widgets such as buttons), their locations and may also describe what happens when an action is executed (e.g. clicking on the hamburger menu in **Figure 4.2-left**). Wireframes can be arranged in a linear sequence, as a sequential storyboard to show a “*visual story of a user experience unfolding over time*” (Greenberg et al., 2011, pp. 151). The visual representations can stop being linear and become state transition diagrams (Greenberg et al., 2011), and eventually become branching storyboards describing the entire usage of the application. User flow refers to the ability of a wireframe to be tested with people and allowing designers to follow through task descriptions. Wireframes play two main roles: (1) outlining the layout



Figure 4.2 Example of a wireframe. The image on the left shows the view of a profile, while the view on the right shows what happens when a user clicks on the top left hamburger menu.

of an application; and (2) defining the user flow (how a user might go from one screen to another following particular tasks). With wireframes, it becomes possible to carry out tests of the design with users to evaluate its usability, and also provide developers with specification of what the user interface should look like as well as what the transitions might be. While nowadays wireframes are primarily done digitally through tools such as Balsamiq or Adobe XD, they can also be created using paper, sticky notes or foam core. Using paper and the like allows for early testing and more flexibility to make changes, while the digital counterpart may work better for specification purposes (e.g. outlining the layout and content to developers).

4.1.3 WIZARD OF OZ

In some cases, it is possible to examine how an experience might play out in a close-to-reality setting even if it is difficult to explore. The Wizard of Oz Technique, first proposed by Kelley (1983), is one way of doing so. Buxton (2007) defines the Wizard of Oz technique as “*making a working system, where the person using it is unaware that some or all of the system’s functions are actually being performed by a human operator, hidden somewhere ‘behind the scene’*” (pp. 240). Buxton describes that the focus is not on the fidelity of the implementation, it is the “*fidelity of the experience*” (pp. 239). As long as the system can appear to be real, it is possible to test different scenarios of how the interaction might play out, and then decide the opportunities and shortcomings to create a more finished design. A good example of Wizard of Oz is an experiment by Gould et al. (1983) at IBM: creating a listening typewriter. The authors used a simulated environment: a

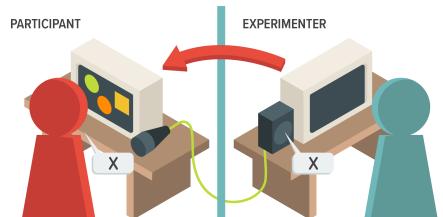


Figure 4.3 Wizard of Oz. Adapted from Gould et al. (1983).

typist was located in another room, listening and typing to a participant dictating to what the participant thought was a regular computer (see **Figure 4.3**). As a result, it was possible to test whether a speech interface for typing would be a sensible idea long before speech recognition algorithms reached such level of sophistication, thus creating a more seamless experience. Gould et al. were able to test different aspects of the listening typewriter, including people's composition time and their preference to this novel approach.

Wizard of Oz excels in the case of testing an envisioned interactive system early on, even if the final implementation might currently be impossible. One challenge, however, is that the human operator (the Wizard) has to constantly be aware of what the participant expects as a behaviour, and realizing the command (or providing cues that the command is invalid). As a result, it is difficult, if not impossible, to provide live feedback, or creating complex dynamic interactions (e.g. simulating a mouse hover state, while still performing other operations). For example, Walny et al. (2012) show how even showing different states on a screen can be a challenge: “*to the study participant, the prototype appears fully functioning, though slow to respond*” (pp. 2870), and “*it was difficult for the wizard to be fully consistent across participants and sometimes even within a participant. In addition to the cognitive load and stress on the wizard to make quick, consistent decisions, the [technology] we used to recognize touch was not 100% reliable*” (pp. 2787). Yet, they still managed to devise and elicit an interaction vocabulary for exploring data visualizations with pen and touch (i.e. post-WIMP interactions), with a level of flexibility of interpretation that might not be possible in a full, robust prototype.

4.1.4 VIDEO PROTOTYPING

In 1988, Mackay (1988) formalized many of the lessons of the Wizard of Oz technique into Video Prototyping. Video prototypes make it so that designers can illustrate interactive and non-interactive demonstrations of software that has not yet been designed. Greenberg et al. (2012) show examples of how video prototypes can be created at a low cost in little time by filming or photographing paper sketches. Thus, video prototypes can work at different fidelities, at the cost of viewers not being able to fully experience what it is like to interact with it first-hand.

4.1.5 PROGRAMMED INTERACTIVE PROTOTYPES

In graphic design, designers often deliberately select design concepts and complete them as a way to further explore and understand problems and potential solutions (Danis and Boies, 2000). Interaction design is no different, and there are instances in which it is necessary to implement a programmed system (or at least a partially functional one) to get a fuller sense of the experience. Lindell (2014) argues that like paper and pencil, code is a malleable material that can be used to explore solutions. Lindell's study shows how some designers describe programming as a means to test their way forward, and that sometimes it feels akin to sketching – and a fundamental tool when an idea is difficult to portray on paper. This view is similar to Myers et al. (2008) where interactive behaviours are reflected as features difficult to describe on paper and would require at least some form of scripting.

Buxton (2007) suggests that the value of developed systems, or “*rapid prototypes*”, is that they afford exploring a class of interaction while providing direct personal experiences, though often limited to lab settings. The programmed prototype can be tried out and in relevant cases further fine-tuned until a behaviour feels right. Lindell (2014) adds that this practice is particularly important if realizing a new interaction technique, where the programming is treated in an exploratory fashion rather than a descriptive manner. Indeed, Interaction designers who are technologically savvy can rely on many tools to solve the problem, and take an opportunistic approach (Brandt et al., 2008), where they combine multiple programming languages and software tools to realize novel interactive technologies.

While seeing a realistic system might seem like the ideal solution, there are a few challenges to consider. From a technical standpoint, programming can be time consuming and require a lot of resources. Moreover, interaction designers are not often literate in programming, which can create more difficulties in the process or simply be inaccessible. This might be why Brandt et al. indicate that opportunistic programming primarily relies on high-level tools with fast iteration and impermanent code (2008). Finally, Buxton (2007) warns against their “*seductive*” qualities, both from an experience standpoint, as well as due to the investment in the system’s creation, which can obscure a critical approach. Later sections of this chapter will discuss means to address the technical challenges, and encourage more explorations – a way to potentially address the challenge of seduction.

4.1.6 HOW DO THESE APPROACHES PROTOTYPE STRUCTURE, BEHAVIOUR AND USAGE?

All of these different prototyping activities yield different benefits. Sketches, given the facility to quickly create them, are a fundamental tool for ideation and for capturing quick thoughts, which is why it is practiced from the onset of the design. Wireframes are a solid way to show specification of the interface – the overall layout and some of the basic interactions (e.g. what happens when a button is pressed). Some systems (e.g. InVision, Adobe XD) also often provide additional features to create animations. Wizard of Oz, video prototyping and interactive programmed prototypes are less common given that they can be time consuming to create, and might even require additional expertise (e.g. ability to edit videos or to program). Wizard of Oz enables simulation and provides some understanding of how a system might communicate with a person, whereas video prototypes provide a third-person envisioning of the interaction. Programmed prototypes are of the highest resolution and require the most effort to create. Yet, from the perspective of behaviour (creating animated transitions and showing in-the-loop behaviours) as well as the perspective of usage (the ability to try them out), programmed prototypes provide the most coverage. Also note that if a prototype has a particular physical form, such as a smart object or appliance, then not many of these approaches cover the experience of interacting with the physical object. Indeed, if a fully interactive programmed prototype has to take place for a smart object, the physical form, or a close representation of it, would likely need to be present as well. **Figure 4.4** summarizes how these approaches contrast to each other. This

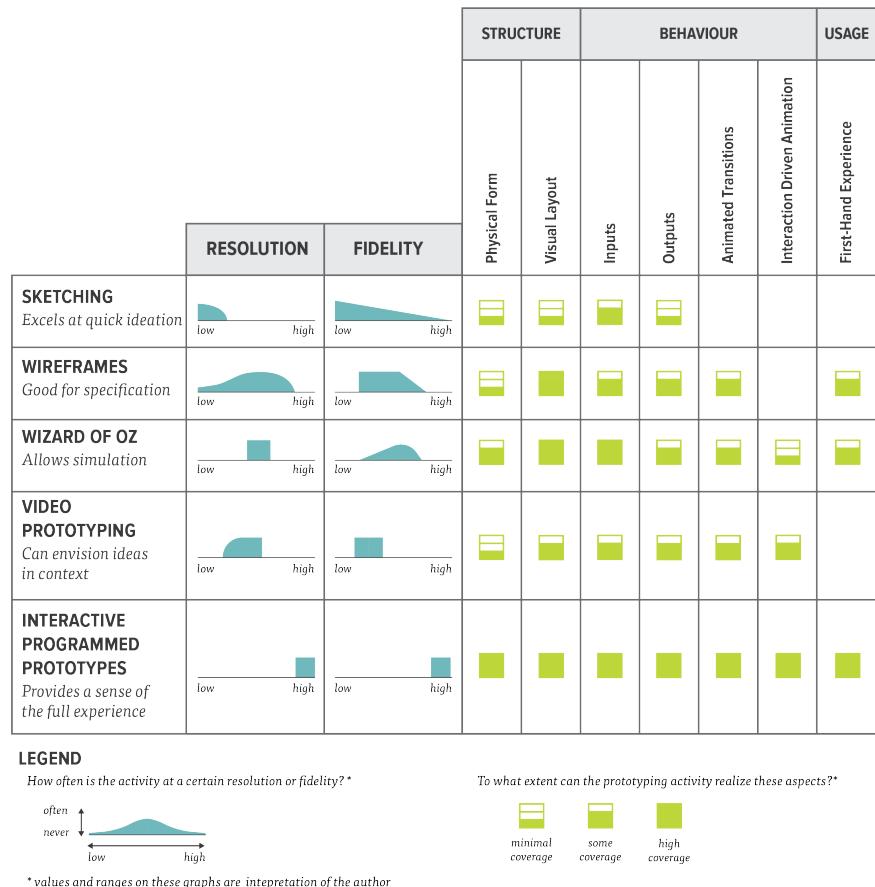


Figure 4.4 Contrasting different prototyping approaches and the extent to which they prototype structure, behaviour and usage elements of an interactive system or artifact. Note how interactive programmed prototypes are of the highest fidelity and resolution yet they also provide the most coverage for what the system might be like.

directly informs the goal of my work of complementing these practices to enable interaction designers to prototype interactive behaviours earlier on and illustrates the current gap on each approach.

4.2 HOW DO DESIGNERS PROTOTYPE INTERACTIVE BEHAVIOUR TODAY? OR DO THEY?

Myers et al. (2008) studied how interaction designers devised interactive behaviours, and found that it was a result of discovery by exploration. Looking back at the different prototyping approaches from

the last section, it would seem like the only suitable approach to try out dynamic behaviour is via programmed interactive prototypes. Interestingly, calls for making prototyping tools that can support a variety of behaviours beyond one-shot animations and that can target more types of devices (e.g. appliances) date back to 1995 (Tscheligi et al., 1995).

Typically, the discussion of interactive behaviour, as explained by Cooper et al. (2014), Saffer (2013), and current prototyping tools such as InVision and Adobe XD, is centred in the following views:

Interactive Behaviour as Screen Transitions. Interactive behaviour can be thought of as elements of an application's flow¹, such as navigating between different screens of the interface, or incorporating user interface widgets and describing their effects.

Interactive Behaviour as Animations. Interactive behaviour can be described as animations and animated transitions that take place in the user interface. These animations can happen as a result of a trigger, or to communicate the current state of an application (e.g. a loading screen).

These two behaviour classifications are important, as they cover a representative portion of the user interfaces today and the authoring necessary for interaction designers. However, this is only a small piece of what interactive behaviour can mean, and with the increased standardization in user interfaces (e.g. Apple Design Do's and

¹ Grigoreanu et al. (2008) define *flow* as a diagrammatic representation to show the structure of the application conveying the flow of data, an analogous term to wireframing.

Don'ts²), designers are only leveraging a limited amount of their skill-set. This becomes even more problematic not only when trying to create applications that move away from the standard, but as new interface paradigms arrive, such as smart objects and head mounted displays.

4.2.1 FORMATIVE INTERVIEWS: CHALLENGES AND NEEDS

There are a few studies in Human–Computer Interaction that discuss interaction designers and the challenges they face when authoring interactive behaviour.

Exploration and Realization. Myers et al. (2008) found key insights with regards to interactive behaviour design: (1) behaviours are more difficult to design than creating the visual layout; (2) behaviours were complex and diverse beyond what a system could provide as built-in behaviours/widgets; (3) behaviours emerge through exploration and fine-tuning (they required iteration). Moreover, designers reflected that interactions needed to be created with details in mind, citing comments such as: “*there are many factors that can influence behaviour*” (pp. 180), and “*there's no such thing as low-fidelity³ interaction, it has to be right*” (pp. 180). Grigoreanu et al. (2008) corroborate these results by investigating interaction designers’ needs, where the needs rated most important ones were “*flow*” and “*feel*”.

² <https://developer.apple.com/design/tips/> – accessed February 2019

³ As per the earlier discussion of fidelity vs. resolution, the designer here likely was referring to resolution.

Communication. The communicative aspect is another challenge in the design of interactive behaviour. Both Myers et al. (2008) and Maudet et al. (2017) share how ultimately developers have to realize the designers' solutions, and the better they can describe it, the more accurate the implementation. Maudet et al. (2017) show how designers will go as far as to create fully animated videos to show how a specific animation or interaction should take place. Video only shows one part of the interaction: the output.

Impossibility. Maudet et al. (2017) show that designers can sometimes generate solutions that cannot be implemented by developers, especially when it comes to custom interactions. Holmquist (2005) posits how given that interaction designers work primarily with software, it should be possible to create representations that can behave close to the product, as "*it can be put into situations that approach those of real use*" (pp. 51). Thus, the issue of impossibility might be a direct consequence of the tools designers use today.

4.2.2 THE (COMMERCIAL) TOOLS DESIGNERS

USE TODAY

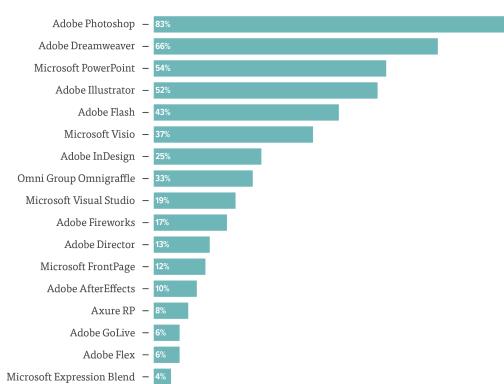
Given the background of interaction designers and what studies share about their practices, reaching a programmed interactive prototype could address some of the challenges in terms of opening current exploration constraints, allowing for accurate communication, and leading to plausible designs. Yet, this vision is challenged by the need for specialization in programming high level behaviours while also doing so in a timely manner. Interaction designers today use a variety of tools to prototype interactive systems. Three online surveys provide a landscape of the current state of the art prototyping tools: Myers et

al. (2008), Subtraction.com (2015), and UXTools.co (Palmer, 2018). **Figure 4.5** shows the results of these surveys. These results reveal a series of insights, discussed next.

Scarcity of Traditional Programming. Aside from a portion of designers doing a variable amount of web development (which can range from HTML and CSS mockups, to Javascript development, to full front-end development), there is no mention of other forms of traditional programming (e.g. C#, java, python, etc.).

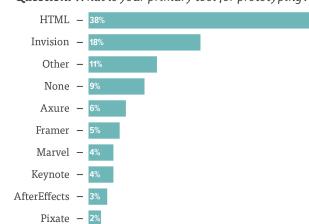
Departure from Authoring + Programming Tools. Related to the last point, it seems there has been a departure from hybrid authoring and programming environments such as Flash, Visual Studio and Expression Blend. These tools allow high-level authoring through pre-built widgets (Visual Studio and Blend), free-form drawing and animation support (Flash and Blend), as well as custom scripting (Flash via ActionScript, Blend and Visual Studio via C#). Perhaps these tools are no longer used due to lack of support (e.g. Flash becoming less prominent on the web), or the emergence of new and more common operating systems (e.g. iOS and Android as opposed to Microsoft-specific alternatives).

MYERS ET. AL (2008) 259 Participants



SUBTRACTION.COM (2015) 4007 Participants

Question: What is your primary tool for prototyping?



UXTOOLS.CO (2018) 2775 Participants

Question: Which tools do you use for prototyping?

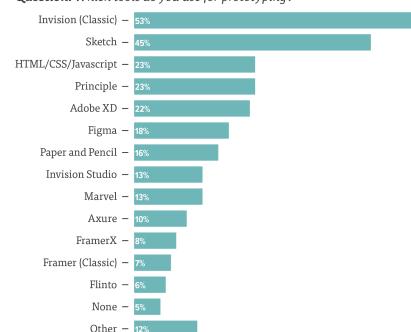


Figure 4.5 Tools interaction designers use today as described by Myers (2008), Subtraction.com (2015) and UXTools.co (2018). Results reproduced from the respective sources.

Focus on Wireframing/Flow. With the exception of AfterEffects and HTML, all the tools in the surveys by Subtraction.com (2015) and UXTools.co (Palmer, 2018) are primarily for wireframing. A few of these tools support some degree of one-shot animations and a few pre-defined triggers (e.g. tap, double tap, scroll) to add further interactivity elements. Thus, it becomes difficult to create custom interactions.

High Standardization. The last aspect common to these prototyping tools of today is the high amount of standardization. It is only possible to author wireframes with the look and feel of standardized web and mobile interfaces, with little room to go beyond or outside of these boundaries. While it is one way of addressing the problem of impossibility, this limits the kinds of experiences designers can create. Myers et al. (2008) unveiled over 100 behaviours designers wanted to create that they were not able to under conventional tools, and yet current tools only support wireframing for standard applications constrained to very simple one-shot animations.

Software Tools Shape How People Think, And What Is Possible

There is a risk of easily dismissing exploration of different prototyping tools in interaction design, given the large number of tools present today. Yet, these tools can play a fundamental role in the envisioning and outlining of current software tools and the tools of tomorrow. There is a direct benefit to exploring the creation of prototyping tools – especially if addressing elements of design not explored before or explored in a limited way (e.g. interactive behaviours), or exploring alternative ways of solving problems. Tools can support a designer's

thinking process, as well as aid them in producing an artifact (Stolterman et al., 2009) – the goal can thus range from “playing” with ideas, to creating artifacts, including the interplay of both. Still, design tools “*will influence what activities the designer sees as important*” (pp. 10). Dalsgaard (2017) explains that tools have an impact on designer practice, as they can shape: perception of the situation, conception of hypotheses, types of externalization, knowledge through action, and possible mediations with other people. Indeed, this is because these types of tools can be seen as a vocabulary and a language (Greenberg, 2007) that provides different *paths of least resistance* (Myers et al., 2000). Indeed, different authoring approaches can provide different mental models, thus the exploration of authoring tools for interaction design can help in finding new types of interaction to support, and different ways of thinking that can best fit each person.

4.3 BEHAVIOUR PROTOTYPING TOOLS IN RESEARCH AND INDUSTRY

Research in Human–Computer Interaction together with different commercial tools in the past and today span across different kinds of approaches which provide a foundation for this thesis. Often times prototyping environments face a trade-off between generalizability and flexibility with complexity and usability, and tools will cater towards different degrees of expertise or learning. Note that all of these approaches involve some degree of *programming* (i.e. breaking down a problem into a set of logical steps), though not all involve *coding* (i.e. programming by writing code). This is an important distinction, as tools will often claim not requiring people to program, while they really mean they devised a programming abstraction alternative to

code. The next sections describe some models for prototyping interactivity, which are not mutually exclusive and often influence each other. The next chapter will dive more in-depth into toolkits and their overall role in research as well as in the evolution of tools and systems of the future.

4.3.1 TRADITIONAL CODING

Traditional coding is the most expressive and flexible way to author interactive systems. This makes sense as it is the basis to all software and hardware applications. While not always accessible to designers, researchers as well as companies have devised different levels of abstraction to facilitate the design of interactive systems. For example, toolkits such as Arduino⁴, Phidgets (Greenberg & Fitchett, 2001) and .NET Gadgeteer (Villar et al., 2013) provide programming support to interface with different hardware components. Arduino does this via a setup and loop model, where two main functions allow providing linear instructions to individual components, or checking their current values. The latter two platforms provide more abstractions in both software (through an object-oriented and event-driven approach) as well as hardware (via custom components that can be connected to the computer, such as sliders, joysticks, etc.).

User interface builders (e.g. Visual Studio, Blend) are one way to allow layout design using predefined and sometimes custom-made widgets.

⁴ Arduino: <http://arduino.cc> – accessed February, 2019.

A. EVENT-DRIVEN PROGRAMMING

```
// global variables
Segment segment;
boolean isDrawing;

void PointerPressed(Point position)
{
    segment.Point1 = position;
    segment.Point2 = position;
}

void PointerMoved(Point position)
{
    if(!isDrawing &&
        Distance(segment.Point1, position) > 3)
    {
        isDrawing = true;
        segment.Point2 = position;
        Draw(segment);
    }
    else
    {
        Erase(segment);
        segment.Point2 = position;
        Draw(segment);
    }
}

void PointerReleased(Point Position)
{
    if(isDrawing)
    {
        Erase(segment);
        // notify application
        OnSelectionCompleted(segment);
    }
}
```

B. INTERACTION MACHINE

```
Interaction RubberBand
{
    Segment segment;

    State Start
    {
        when PointerPressed(Point position)
        {
            segment.Point1 = position;
            segment.Point2 = position;
        } -> WaitMove
    }

    State WaitMove
    {
        when PointedMoved(Point position),
        && Distance(segment.Point1, position) > 3
        {
            segment.Point2 = position;
            Draw(segment);
        }
        when PointerReleased -> Start
    }

    State Track
    {
        when PointerMoved(Point position)
        {
            Erase(segment);
            segment.Point2 = position;
            Draw(segment)
        }
        when PointerReleased
        {
            Erase(segment);
            OnSelectionCompleted(segment)
        } -> Start
    }
}
```

Figure 4.6 Event model vs. interaction machine. Beaudouin-Lafon (2004) exemplifies the contrast between (a) event-driven programming and (b) an interaction machine to implement a rubber-band selection. A rubber-band selection consists of allowing the user to draw a free-form area and select the contents inside of it (e.g. shapes, text, images). Figure recreated from Beaudouin-Lafon (2004).

These widgets (e.g. buttons, checkboxes, sliders) can later be accessed in code, and subscribe to events (e.g. click, pointer⁵ pressed, pointer enter, pointer leave) to provide different behaviours. Often times these interface builders are constrained to a particular platform, though systems such as Gummy (Meskens et al., 2008) and Gummy Live (Meskens et al., 2009) examine models to make the interface building generalize across different platforms. In fact, Gummy Live (Meskens et al., 2009) allows dynamic rendering of the user interface

⁵ Pointer often refers to different input devices such as mouse, pen, or touch.

on the target device (e.g. phone) as it is being drawn on the main computer.

While interface builders are powerful in providing the presentation of the interface, Beaudouin-Lafon (2004) contests that they provide basic interaction, but do not support direct manipulation techniques (e.g. rubber-band selection). Beaudouin-Lafon adds that to create these interactions, developers are forced to “*resort to tricks such as global variables and unsafe narrowing to share state between chunks*” (pp. 20), resulting in “*brittle code that is hard to debug and hard to maintain*” (pp. 20). In this discussion, Beaudouin-Lafon shows how even programming is already far-away from supporting the design of interactive systems, exemplified in **Figure 4.6**.

4.3.2 VISUAL PROGRAMMING

One way to abstract programming into a way that is more accessible is via visual programming. There are two primary ways that visual programming has been carried out in the past: node-link diagrams and blocks.

The idea behind *node-link diagrams* is to treat programs as a flow of data which gets converted until reaching an output, where authors can visually inspect how the inputs are being transformed. Ko et al. (2004) describe “data flow” as an abstract approach that remains human-centric. Max/MSP⁶, shown in **Figure 4.7 A and B**, is an authoring software for interactive sounds and graphics (originally created for music, but later extended to support other areas such as custom

⁶ <https://cycling74.com/products/max/> – accessed February, 2019

electronics and video). One interesting feature of Max/MSP is the inclusion of custom widgets (e.g. piano keys) to make more sense of the flow of data. Nintendo Labo's Toy-Con Garage⁷ takes a similar approach (albeit more simplified) to support the creation of “inventions” using the Nintendo Switch controllers and their sensors/outputs to author interactive behaviours, such as making a controller vibrate on a physical shake action. Trigger-Action Circuits (Anderson et al., 2017) also leverages node-link diagrams to automatically generate multiple circuit diagram alternatives, with assembly instructions and firmware code. Node-link diagrams have the advantage of supporting one-off experiences, with three noticeable drawbacks. First, the data-flow model emphasizes either discrete one-shot actions (e.g. when a piano key is pressed, play a sound) or input transformations that may not be immediately obvious (e.g. making an input, such as the sound average frequency, affect the colour settings of a video feed). Second, there is a strong reliance on prebuilt black boxes that define what effect they will apply to the input. The last problem is scale, as writing complex programs will lead to cluttered screens with many scattered nodes and links that appear messy and have very low readability.

A more recent approach to *visual programming* is the use of *visual blocks*, which improve readability and usability via a linear structure. With the development of Scratch (Maloney et al., 2010), shown in **Figure 4.7 C**, different kinds of programming constructs (commands, functions, triggers and control) have a structure with visual notches

⁷ <https://labo.nintendo.com/invent/> – accessed February, 2019

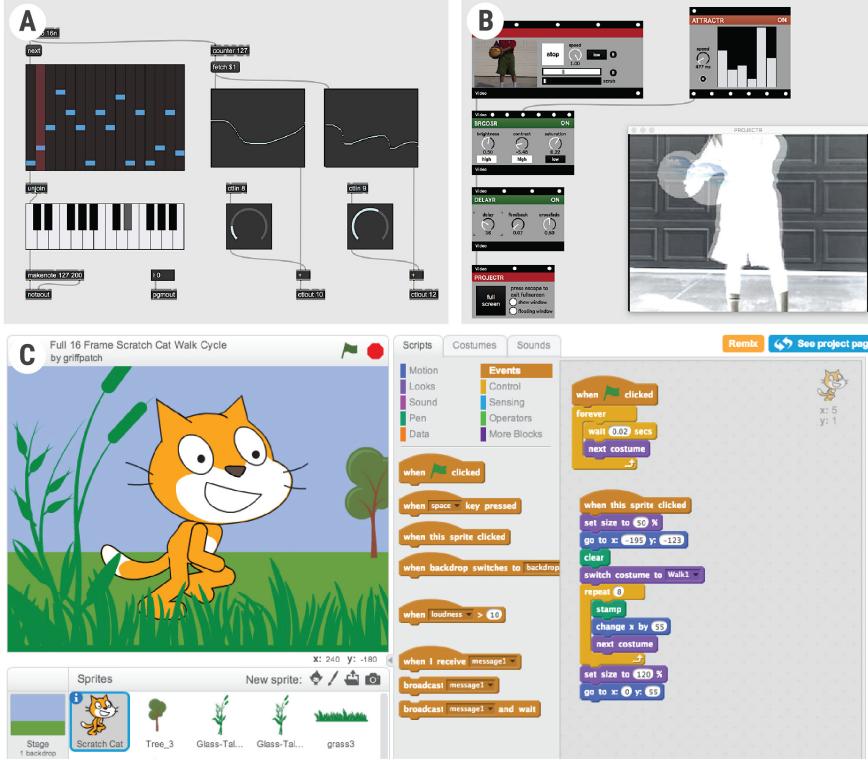


Figure 4.7 Examples of visual programming approaches. (A) and (B) show node-link diagrams Max/MSP, where (A) performs audio mappings, while (B) modifies a video feed. (C) Shows Scratch, a block-based language. (A) and (B) taken from <https://cy-cling74.com/products/max/>, while (C) is taken from <https://www.aace.org/review/prepare-for-fun-scratch-3-0-is-coming/>

and exposed parameters. Scratch has been adapted into different contexts, including Microsoft MakeCode⁸ which can leverage the scratch platform to operate electronics platforms including Arduino and Microbit⁹.

4.3.3 SCREEN TRANSITIONS

A natural software parallel to wire-framing includes the family of systems which feature screen transitions and some degree of custom

⁸ <https://www.microsoft.com/en-us/makecode> – accessed February, 2019

⁹ <https://microbit.org/> – accessed February, 2019

functionality. HyperCard by Apple Computer shows abstractions of text, graphics, multi-media objects (e.g. sounds and video), as well as user interface widgets into “cards” which could be interconnected and linked (Nielsen et al., 1991). Objects could be drawn on screen similar to a traditional graphical user interface builder, and scripts could link the functionality from one card to the next. These cards¹⁰ would become integrated within the larger HyperCard program. Goodman (1988) describes the target audience (which he calls “developers”) as computer consultants that create information tools for their clients (e.g. a kiosk), teachers writing simulations for other students, etc. Thus, the goal was for the system to be accessible to a more casual set of users who could create different kinds of interactive multimedia presentations. Nielsen et al. (1991) describe how HyperCard could be used for prototyping graphical user interfaces. In later years, this would be a process taken by presentation tools such as Microsoft PowerPoint and Apple Keynote, as demonstrated by Greenberg et al. (2012).

The challenge with using presentation software for prototyping is that it worked as a re-appropriation for tools that were not designed for that purpose. This changed with systems such as SILK (Landay, 1996), DENIM (Lin et al., 2000) and DEMAIS (Bailey et al., 2001) which leverage sketching as a way to draw interfaces and create connections between widgets and screens. In particular, DEMAIS emphasized visibility and discoverability through a visual language that

¹⁰ While documentation of HyperCard is scarce, online resources offer an introductory video (<https://www.youtube.com/watch?v=EMFscTOazS0>) and a collection of cards (<https://archive.org/details/hypercardstacks>) – accessed February, 2019

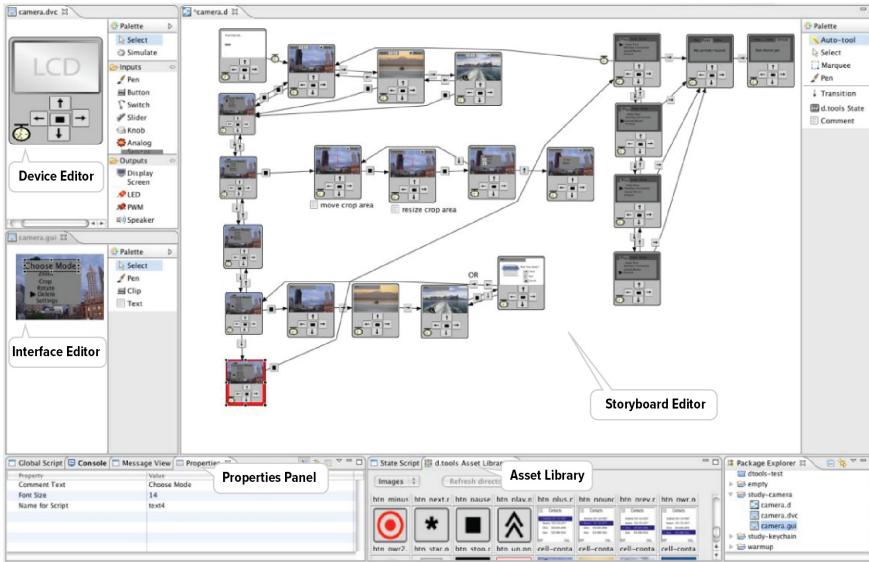


Figure 4.8 The d.tools interface (annotated). Image reproduced and modified from Hartmann (2009).

summarized the different transitions and effects. Subsequently, d.tools (Hartmann et al., 2006), shown in Figure 4.8, enabled novel applications in the area of physical computing could be prototyped through the use of state machine diagrams¹¹. These diagrams made it so designers could connect different states and transition from one to another through input triggers (e.g. switching between images on an LCD display using accelerometer values). While these transitions were discrete, it was possible to connect the system to a Java backend and code continuous transitions (which Hartmann et al. describe as opportunities to collaborate with developers). Many of the commercial prototyping tools today, such as Adobe XD, InVision, Figma,

¹¹ State transitions are talked about in two ways in Human–Computer Interaction literature. One is screen transitions, where one input (e.g. button click) switches the screen to another point. The other is Buxton’s description of input (1990), where the state transition refers to the global state (sometimes referred to as *mode*) of the system based on input.

Framer, etc. follow the state transition pattern, though are limited to web or standard mobile interfaces.

4.3.4 TIMELINE

Another common approach to designing behaviour emerged from animation and video editing – the use of the *timeline*. Graphical representations of time for animation date back to the late 1960's, referred to as “*Picture-Driven Animation*” (Baecker, 1969). In particular, Adobe Flash (formerly Macromedia Flash) brought forward three main features. The first was the concept of *motion tweens*, where animators could create a transition for an object across two keyframes (e.g. changing size, position or colour), and the system would automatically interpolate between them in a linear fashion, creating smooth animated transitions (see **Figure 4.9**). These tweens could be further customized through easing functions with different mathematical operations (Penner, 2002). The second innovation was the introduction of *Actionscript* in which it became possible to write code and logic and associate it to a particular keyframe. The last innovation was the ability to group drawings in a scene into self-contained objects (e.g. movie clips), which meant that even if the animation was paused, the objects could – via scripting – change between different animations. This means that for example, a platforming game could feature a character object that displays motion animations and moves around when an arrow key is being pressed. Thus, one could create interactive environments since different scene objects could behave independently from each other without being bound to the main timeline. Blend adopted some of the ideas of the Flash timeline, where a

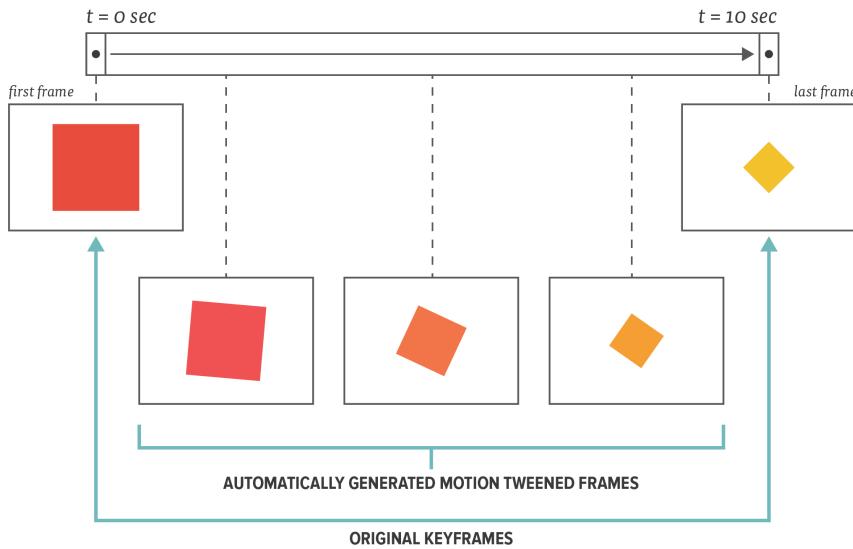


Figure 4.9 How motion tweening works. The animator draws the first frame, and applies transformations to the last frame (e.g. changing fill, size and orientation). The system automatically interpolates between the two frames to create a continuous animation.

group of objects are encapsulated into “storyboards” and the storyboard can be connected to a trigger (e.g. play storyboard when clicking a button) without the need to any write code.

4.3.5 PROGRAMMING BY EXAMPLE

Programming by Example, also known as Programming by Demonstration, is a common approach to create programming environments. Halbert (1984) defines programming by example as taking a user perspective – “*the statements in [their] program are the same as the commands [they] would normally give the system*” (pp. 4) and the program “*is written by remembering what the user does*”. As a result, the collection of demonstrated cases (both positive and negative examples) define what should happen. One early example of programming by example is Topaz (Myers, 1998), which uses the command pattern to record scripts. Myers (1986) recognized a main limitation of the

demonstration approach in its ability to generalize, as the “*user provides no guidance about the structure of the program so each new example can radically change the program. The programmer often knows... which values are variables and which are constants, or where the conditionals should go, but there is no way to directly convey this information to these systems*” (pp. 64). Consequently, systems opt to only use programming by example in a partial manner, or resort to other constraints (e.g. pattern matching) which might limit what the program can do.

In 1994, Click Team’s Klik and Play¹² applied a mix of demonstration and questions to enable people to create games. A character could be manipulated to teach it a motion pattern it should pursue, and running the game would prompt questions as events took place (e.g. “*person and enemy have collided*”, or “*user pressed the spacebar*”) and provide a set of possible predefined commands (e.g. “*subtract a life*”, “*make character jump*”).

Demonstrations are an effective way to distinguish discrete patterns in individual sensor data. In A CAPella (Dey et al., 2004), one could design context-aware applications by demonstrations based on discrete data from multiple sensors, a camera and microphone. With Exemplar (Hartmann et al., 2007), a designer could take an individual sensor with discrete or continuous data (e.g. an accelerometer) and match a pattern (e.g. shaking) or a threshold crossing (e.g. acceleration in X > 5) to define a discrete action. The viewing of sensor data

¹² Information about Klik and Play is scarce, but a video demonstration can be found in <https://youtu.be/LUTpumYboDs> – accessed February 2019.

took place in real time. Similarly, Sauron (Savage et al., 2013) leveraged a camera inside a 3D print to identify actions on physical widgets (e.g. moving a joystick). Note that these tools stop at the recognition stage and offload the interactivity design to other software – d.Tools in the case of Exemplar, and OSC sockets in the case of Sauron.

Programming by example has also led to the recording of macros as a way to creating custom interactions. For example, D-Macs (Meskens et al., 2010) leveraged demonstration atop of a GUI designer to create multi-device interfaces, allowing the recording of screens and actions on a desktop and then replaying the actions from on the mobile interfaces. Similarly, Sugilite (Li et al., 2017) allows creating complex multimodal actions on mobile interfaces.

Programming by demonstration provides a means to create commands via direct manipulation, it excels at capturing potential triggers to create a reactive interactive behaviour.

4.3.6 KEYFRAMING

A few systems have taken the programming by demonstration approach in combination with the motion tweens provided in systems such as Adobe Flash. Monet (Li & Landay, 2005) allowed designers to manipulate objects and define the point of input, allowing the authoring of different interactions such as sliding a mouse across a dial or scaling an object using the mouse cursor. Similarly, different object animations could also be linked between each other. In Monet it is possible to create different interactive behaviours for custom widgets and mouse-based interactions, such as a scroll bar, drag and drop, etc. Systems such as Kitty (Habib et al., 2014), and Espresso (Krosnick

et al., 2018) apply these concepts in different contexts. In particular, Espresso allows creating responsive web designs that will change as a window is resized, thus one can map the position of visual objects to the size of the current browser window.

4.3.7 STAGE METAPHOR

Finzer and Gould (1993) created a programming environment made for non-programmers to create education software based on the metaphor of theatre: “*only things that can be seen can be manipulated*” (pp. 1). In that sense, the environment had *performers*, *stages*, and interacted via *cues* to each other. Adobe Director (formerly Macromedia) leveraged this stage metaphor and provided additional scripting via Lingo. Given Director’s built-in 3D engine and ability to support more complex programming, toolkits such as DART (MacIntyre et al., 2004) could support authoring Augmented Reality applications. YoYo Games’ Game Maker contextualizes these principles into game design, where people can create characters that have a set of visuals (i.e. sprites) and attach behaviours to them based on events (e.g. when the screen refreshes, when a key is pressed) via scripts or node-link visual programs. Note that all these approaches require some degree of coding even if in the background.

4.3.8 STEP BY STEP WIZARDS

Perhaps one way to eliminate the programming gap is to make the task less about programming and more about configuration. One can create a system that forces (1) following steps in a (2) particular order to accomplish a goal, where (3) different parameters can be set. This

is often referred to as the *Wizard Pattern*¹³, inspired by software installation “wizards”. PYGMALION (Smith, 1975) is perhaps the first system to do this, emphasizing “*doing rather than telling*” (pp. 68). Several systems support this paradigm in different degrees. IFTTT¹⁴ allows connecting multiple web services (e.g. “*turn on the lights when the pizza delivery arrives*”) via pre-built trigger-action connections. Midas (Savage et al., 2012) provided customization of capacitive sensors to re-route touch events which could be interpreted by custom microcontrollers as well as mobile devices. In Midas, designers lay out touch sensitive areas on a mobile device image to create a fabrication-ready circuit, and map those touch-points to pre-recorded actions or as WebSocket events that can be picked up by other applications. PaperPulse (Ramakers et al., 2014) similarly supported creating interactive paper-based circuits, where the software first provided a widget builder which then allowed recording of custom trigger-action events. Similarly, RetroFab (Ramakers et al., 2015) made appliances smart by adding custom electronic components which could then have actions recorded to define the behaviours and provide controls from a mobile application.

4.3.9 WIZARD OF OZ AND VIDEO PROTOTYPING TOOLS

Some tools have looked to support Wizard of Oz and video prototyping practices. In particular, de Sá et al. (2008) show how one can use

¹³ Nick Babich (2017) – Wizard Design Pattern <https://uxplanet.org/wizard-design-pattern-8c86e14f2a38> – accessed February, 2019.

¹⁴ IFTTT <https://ifttt.com/> – accessed February, 2019.

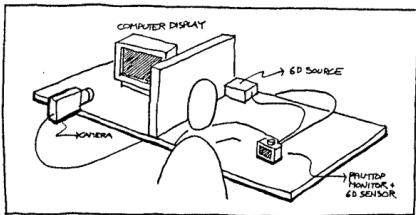


Figure 4.10 Setup for Chameleon by Fitzmaurice (1993). Setup shows how a portable TV augmented with a button and position sensors is rendering the video-streamed images from a camera pointed at a monitor which runs the software applications for mobile spatial navigations. Image reproduced from Fitzmaurice (1993).

photos of sketched or rendered wireframes on mobile devices to simulate an interactive system. It is worth noting how de Sá et al. reveal how seeing the sketches in context detected usability issues early on, but more importantly, seeing the interface on the target screen made people realize early on when controls were inadequate, or sizing (e.g. amount of text) was an issue. ProtoAR (Nebeling et al., 2018) allows designers to place overlays (e.g. sketches) atop clay models captured with a mobile camera, thus allowing prototyping of Augmented Reality applications. Montage by Leiva and Beaudouin-Lafon (2018) uses (1) a user camera capturing a scene with a context containing a green screen, (2) a wizard camera directed at a paper prototype, and (3) a tablet canvas which places the paper prototype atop the green screen using a chroma-key technique. Through this, they can record video and use a timeline to change the contents of the interface. What is interesting about these Wizard of Oz tools is in their adoption of some of the aforementioned approaches (e.g. the timeline in Montage). While some of these tools do support live rendering, they remain as simulations which do not allow testing of the overall experience.

4.3.10 SMOKE AND MIRRORS AND SCREEN POKING

Buxton (2007) describes an alternative way of achieving high fidelity experiences, which he labels “*smoke-and-mirrors*” technologies. Instead of relying on a human operator as in Wizard of Oz, designers realize an interactive sketch of a concept through “*clever use of technologies and techniques*” (pp. 245). Buxton exemplifies this approach through Fitzmaurice’s work on Chameleon (1993). Chameleon simulates a mobile device that is spatially aware, and was realized through the following setup shown in **Figure 4.10**: the end-user holds

handheld portable TV with a small motion capture device attached to the back; the sensors are connected to a computer running a three-dimensional map application rendered on a display; and a video camera reflects the visuals of the display on the handheld TV. From the end-user's perspective, they were holding a mobile device able to sense spatial interactions at a time in which palmtop computers (and further with such graphics capabilities) were not readily available. While Fitzmaurice shows a hardware workaround to simulate highly capable mobile interactions, the software still required implementation.

The last example is a very sophisticated prototype, which is not a necessary requirement for smoke and mirror prototypes. A simple, yet still powerful approach, is to repurpose the mouse and keyboard events within an application in an interesting way, a technique which Hartmann (2009) calls "*screen poking*". An example of this is Leganchuk's Doorstates (Buxton, 1997), where a physical door was used as a means to communicate accessibility in video conferencing by repurposing a mouse mounted by the door. Hudson and Mankoff (2006) created BOXES, a hardware platform to create physical prototypes out of cardboard, which then connected to an application, Thumbtacks, to assign mouse and keyboard events or recordings to control familiar applications, to create simple hardware prototypes that could behave in more sophisticated ways (e.g. tapping a piece of cardboard would play music by clicking on the media player's play button). Makey Makey¹⁵ created a similar platform, albeit more simplified, where

¹⁵ Makey Makey <https://makeymakey.com/> – accessed February, 2019.

one could connect alligator keys to a custom board and the board automatically maps those events to mouse and arrow keys or mouse button events. Hartmann (2009) also created functions to support screen poking within d.Tools, where he describes that the limitation of screen poking is that it is “*unaware of the internal state of the controlled application*” (pp. 87). Thus, while it is possible to use any application, designers need to consider some degree of setup of the computer application, and recognize that there may not always be full control of the external application.

4.4 SUMMARY AND CONCLUSION

After discussing different prototyping activities carried out by designers (e.g. sketching, wireframing), it becomes clear that designers can only truly explore behaviours and physically try them out by creating interactive programmed prototypes. Thus, I identify a gap in current practices and in the commonly used commercial tools. Prior systems tackle some of these elements, which directly inform my work in providing means to author nuanced interactive behaviours for smart interactive objects. Seeing the existing gap, and noting the added complexities of smart object prototyping, discussed in Chapter 6, this collective work informed the design rationale that led to the research contributions in Part 2 and Part 3 of this dissertation.

The knowledge described in Chapters 2, 3 and 4, provides a context on the skills and expectations of interaction designers, along with their common practices. Different tools were depicted to show authoring approaches which can be complemented with alternative solutions to (1) support exploration and behaviour authoring, (2) allow the creation of smart objects, and (3) foster new ways of thinking

about how design tools can guide designers in the creation process. Moreover, another element that has provided further reflection and insight is in having an understanding of the work done in HCI toolkits and more importantly how these authoring systems are evaluated. Understanding evaluation methods in advance helped better shape the design of prototyping tools, as it enables thinking about what aspects to support and how to communicate them, which is the topic of the next chapter.

CHAPTER 5. EVALUATING TOOLKIT SYSTEMS

This chapter describes evaluation methods used in HCI toolkit research¹. At first glance, this chapter might appear secondary to the dissertation’s technical goals of producing toolkits²/prototyping tools for interaction designers. Yet, toolkit evaluation – and the decisions made around it are central to any academic discussion of toolkit design. Many challenges accompany toolkit evaluation (§5.1), and there is no consensus on whether it should be done, and if so, how. The research portrayed in this chapter is an attempt to appraise and

¹ Portions of this chapter have been published in:

ledo, D*. Houben, S.* Vermeulen, J.* Marquardt, N., Oehlberg, L., & Greenberg, S. (2018). Evaluation Strategies for HCI Toolkit Research. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–17. doi: [10.1145/3173574.3173610](https://doi.org/10.1145/3173574.3173610)

* Authors contributed equally to the work.

Data and other materials can be found at: <https://github.com/davidledo/toolkit-evaluation>.

² Prototyping tools in the context of interactive behaviour design fall into the definition of toolkits (§5.2). I argue that these prototyping tools are a subclass of toolkits, and their building blocks operate as programming tools that do not require writing code.

critically discuss (§5.2) the role of evaluation in toolkit research, which in turn lays the foundation for why particular decisions were made in evaluating the tools described in this dissertation.

Based on an analysis of 68 representative toolkit papers (§5.3), this chapter contributes an overview and in-depth discussion of evaluation methods for toolkits in HCI research. The survey resulted in four types of evaluation strategies: (1) *demonstration* (§5.4), (2) *usage* (§5.5), (3) *technical benchmarks* (§5.6), and (4) *heuristics* (§5.7). I present these four evaluation types, and opine on the value and limitations associated with each strategy. This meta-review synthesis is based on a sample of representative toolkit papers. Further, I link interpretations to both our own experiences as toolkit authors, and earlier work by other toolkit researchers (§5.8). Researchers can use this synthesis of methods to consider and select appropriate evaluation techniques for their toolkit research.

5.1 THE CHALLENGE OF TOOLKIT EVALUATION

Within HCI, Greenberg (2007) defined toolkits as a way to encapsulate interface design concepts for programmers, including widget sets, interface builders, and development environments. Such toolkits are used by designers and developers to create interactive applications. Thus, they are generative platforms designed to create new artifacts, while simplifying the authoring process and enabling creative exploration.

While toolkits in HCI research are widespread, researchers experience toolkit papers as being hard to publish (Nebeling, 2017) for various reasons. For example, toolkits are sometimes considered as

merely engineering, as opposed to research, when in reality some interactive systems are ‘sketches’ using code as a medium to explore research contributions, whereas others embody their contributions in the code itself (Fogarty, 2017). Sometimes, toolkit researchers are asked for a particular evaluation method without consideration of whether such an evaluation is necessary or appropriate to the particular toolkit contribution. Consequently, acceptance of toolkits as a research contribution remains a challenge and a topic of much recurrent discussion, such as Bernstein et. al (2011), Fogarty (2017), Greenberg (2007), Marquardt et. al (2017), Myers et. al (2000) and Olsen (2007). In line with other areas of HCI (Greenberg, 2007; Olsen, 2007), we should expect HCI toolkit research to use appropriate evaluation methods to best match the particular research problem under consideration, as discussed by Greenberg and Buxton (2008), Hudson and Mankoff (2014), and Preece and Rombach (1994). However, while research to date has used different evaluation methods, there is little overall reflection on *what* methods are used to evaluate toolkits, *when* these are appropriate, and *how* the methods achieve this through different techniques.

The last two decades have seen an increase in HCI toolkit papers (Marquardt et. al, 2017). These papers typically employ a range of evaluation methods, often borrowing and combining techniques from software engineering, design, and usability evaluation. From this corpus, it is possible to derive *what* evaluation methods are useful, *when* they are appropriate and *how* they are performed.

5.2 WHAT IS A TOOLKIT?

Within HCI literature, the term ‘toolkit’ is widely used to describe various types of software, hardware, design and conceptual frameworks. Toolkit research falls into a category of constructive research, which Oulasvirta and Hornbæk (2016) define as “*producing understanding about the construction of an interactive artefact for some purpose in human use of computing*” (pp. 4958). They specify that constructive research is driven by the absence of a (full) known solution or resources to implement and deploy that solution.

As constructive research, toolkits examine new conceptual, design or technical solutions to unsolved problems. To clarify this chapter’s review’s scope, I next define and summarize what is meant by “*toolkit*” and “*toolkit evaluation*”, and why HCI researchers build toolkits.

5.2.1 DEFINING A TOOLKIT

I extend Greenberg’s original definition (2007) to define toolkits as *generative platforms designed to create new interactive artifacts, provide easy access to complex algorithms, enable fast prototyping of software and hardware interfaces, and/or enable creative exploration of design spaces*. Hence, toolkits present users with a programming or configuration environment consisting of many defined permutable building blocks, structures, or primitives, with a sequencing of logical or design flow affording a path of least resistance. Toolkits may include automation (e.g. recognizing and saving gestures (Marquardt et. al, 2011a)) or monitoring real-time data (e.g. visualization tools (Marquardt et. al, 2011b)) to provide developers with information about their own process and results.

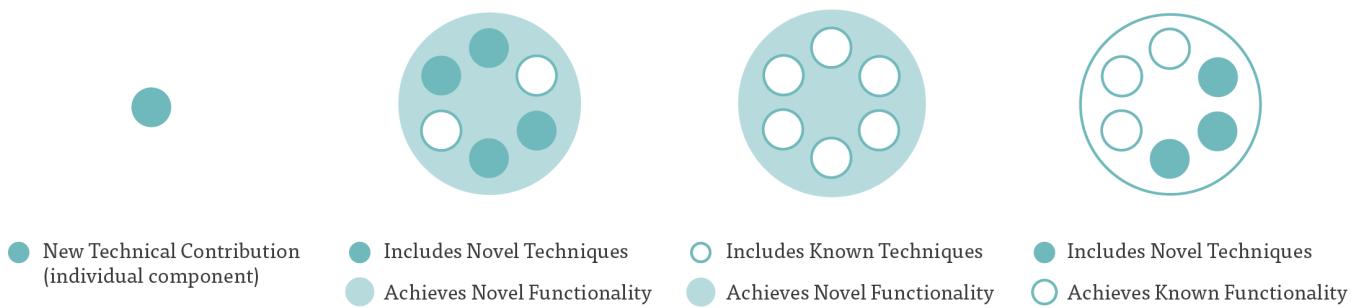


Figure 5.1 Code and contribution as described by Fogarty (2017). Figure illustrates how system contributions are described within HCI research: stand-alone novel technical contribution; a combination of novel and known techniques to achieve novel functionality; using known techniques to achieve novel functionality; or achieving known functionality with novel techniques. Figured adapted from (Fogarty, 2017).

5.2.2 WHY DO HCI RESEARCHERS BUILD TOOLKITS?

Before discussing toolkit evaluation, we elaborate on what they contribute to HCI research. Wobbrock and Kientz position toolkits as *artifact* contributions, where “*new knowledge is embedded in and manifested by artifacts and the supporting materials that describe them*” (pp. 40). Discussions by Myers et. al (2000), Olsen (2007) and Greenberg (2007) on the *value* of HCI toolkits can be summarized into five goals:

G1. Reducing Authoring Time and Complexity. Toolkits make it easier for users to author new interactive systems by encapsulating concepts to simplify expertise (Greenberg, 2007; Olsen, 2007).

G2. Creating Paths of Least Resistance. Toolkits define rules or pathways for users to create new solutions, leading them to right solutions and away from wrong ones (Myers et. al, 2000).

G3. Empowering New Audiences. Given that toolkits reduce the effort to build new interactive solutions, they can enable new audiences to author these solutions. For example, Olsen (2007) discusses how interface builders opened interface design to artists and designers.

G4. Integrating with Current Practices and Infrastructures.

Toolkits can align their ideas to existing infrastructure and standards, enabling power in combination (Olsen, 2007) and highlighting the value of infrastructure research for HCI (Edwards et. al, 2010). For example, D3 (Bostock et. al, 2011) integrated with popular existing standards, which arguably contributed significantly to its uptake.

G5. Enabling Replication and Creative Exploration. Toolkits allow for replication of ideas that explore a concept (Greenberg, 2007), which collectively can create a new suite of tools that work together to enable scale and create “*larger [and] more powerful solutions than ever before*” (Olsen, 2007; pp. 252).

Toolkits serve different roles in terms of their research contribution. Fogarty (2017), as illustrated in **Figure 5.1**, examines where the novelty of a system lies. He suggests that in contrast to an individual technical contribution, a toolkit is a collection of techniques that can achieve a particular functionality, where the functionality is the goal (the “*what*”) and the technique collection describes “*how*” it is achieved. The techniques and functionality can stand as a research contribution in three ways: (1) a toolkit achieves a novel functionality via novel techniques, (2) a toolkit achieves novel functionality through known techniques (e.g. through combination of known approaches), or (3) a toolkit achieves known functionality through novel techniques (e.g. using more optimized algorithms). Greenberg (2007), focuses more on toolkits as a mean of promoting replication by applying Gaines’ (1991) BRETAM model to forecast information sciences (**Figure 5.2**). The model discusses different stages of the

adoption of technology, in which ideas are born through major breakthroughs, which are then replicated. Overtime, the concepts reach empiricism which are formalized into theories. These theories are then accepted and used to predict experiences until those theories are assimilated and used without question. Greenberg's replication is not constrained to recreating a system to achieve the same result. Instead, it is about being able to explore different facets of an idea, such as the application scenarios, or different variations, to reach a richer understanding of the research vision. Under this vision, toolkits are instrumental in supporting more researchers to explore these different application areas.

Marquardt et. al (2017) see promise in toolkits as a research methodology within HCI. First, toolkits embody a generative means to realize theoretical frameworks (e.g. Rogers (2004) and Wiberg and Stolterman (2014)). For instance, the Proximity Toolkit (Marquardt

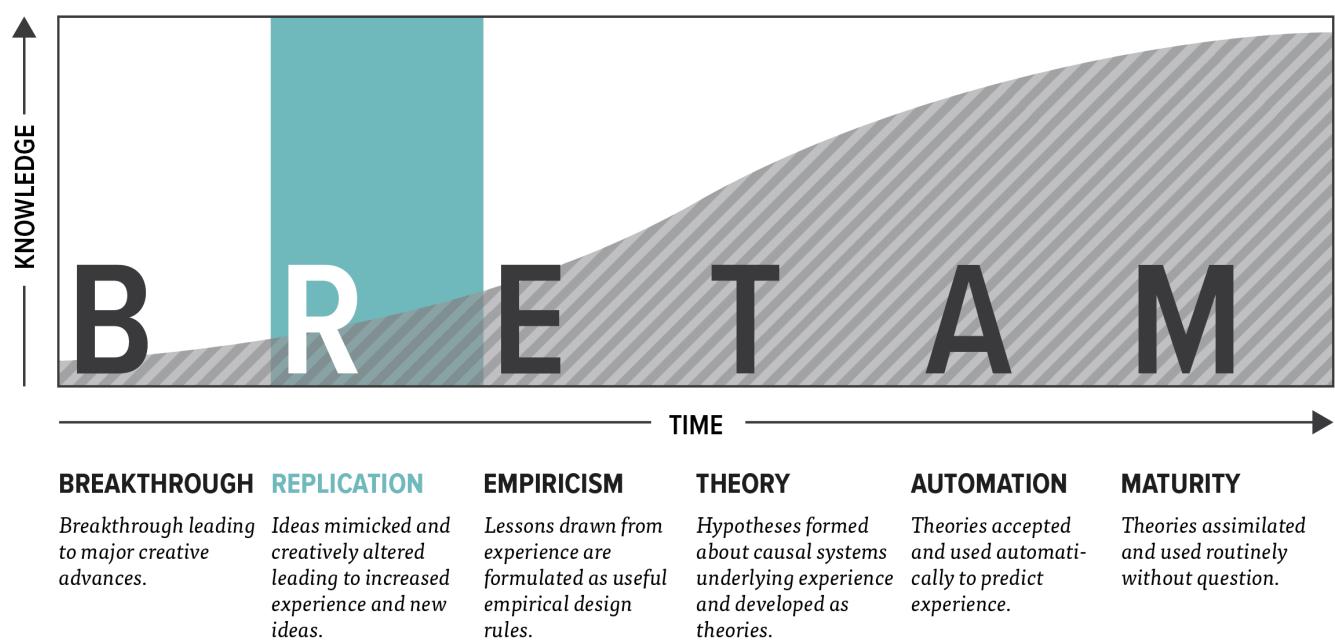


Figure 5.2 Gaines' (1991) BRETAM model of forecasting information sciences as described by Greenberg (2007). Within it, toolkits foster replication to aid design exploration (Adapted from Greenberg (2007)).

et. al, 2011) encapsulated the conceptual building blocks proposed in Proxemic Interaction (Ballendat et. al, 2010), which then fostered more focused research applications in multiple areas, including devising design patterns (Marquardt et. al, 2012), advertising (Wang et. al, 2012), body-centric interaction (Chen et. al, 2012), awareness of shoulder-surfing in public displays (Brudy et. al, 2014), remote controls (Ledo et. al, 2015), etc. Besides applying the Ballendat et al.'s (2010) framework for proxemic interaction, these research endeavours helped further explore what is possible in the research domain while removing much of the programming complexity (e.g. tracking people and devices in 3D space, calculating physical relationships between people and devices). Second, methodologies in design research (e.g. (Cross, 1999), (Hevner et. al, 2004), (Zimmerman, 2007)) suggest that the design of artifacts provide researchers with an understanding of the solution space, while the artifact, in this case, the toolkit, is an embodiment of the knowledge. With toolkits, researchers can experiment creating different prototypes, and in that process can gain a better understanding of (1) *the design space*, given that in the creation process, the building blocks can be immediately reconfigured to generate new solutions, thus further expanding the understanding of the design space; and (2) *the paths of least resistance* to support the authoring practice, as the toolkit creation process implies adapting the authoring process to support different solutions and approaches in a generalizable way. Under this view, the toolkit can also act as a means of exploration, as researchers can devise new ways of authoring technologies to generate multiple solutions. Thus, the creation of a toolkit can be considered a *malleable* process that is informed by an understanding of the target audience, a set of sketched

ideas, and the continued realization of individual prototypes while considering how the authoring approach might generalize to other variations or scenarios.

5.2.3 EVALUATING TOOLKITS

A common concern among HCI toolkit and system researchers is the difficulty in publishing (Nebeling, 2017). This might be due to the expectations and prevalence of evaluation methods (e.g. user studies), regardless of whether the methods are necessary or appropriate to the toolkit's contribution. Part of the problem is a lack of clear methods (Nebeling, 2017) or a clear definition of 'evaluation' within a toolkit context. My stance in this dissertation is that the evaluation of a toolkit *must* stem from the toolkit designer's claims. Evaluation is a means to follow through with the proposed claims of the innovation. Toolkit designers should thus ask themselves: "*what do we get out of the evaluation?*"

Toolkits are typically different from systems that perform one task (e.g. a system, algorithm, or an interaction technique) as they provide generative, open-ended authoring within a solution space. Toolkit users can create different solutions by reusing, combining and adapting the building blocks provided by the toolkit. Consequently, the trade-off to such generative power is the large space that remains under explored. Evaluation methods that only examine a small subset of the toolkit may not demonstrate the research contribution, nor do they necessarily determine a toolkit's success. As summarized by Olsen (2007) in his reflective paper on evaluating systems research: "*simple metrics can produce simplistic progress that is not necessarily meaningful.*"

The central question is thus: *what is an evaluation?* And, *how do we reflect and evaluate such complex toolkit research?*

5.3 METHODOLOGY

This chapter elucidates evaluation practices observed in modern toolkit research within the HCI community. To build up an in-depth understanding of contemporary evaluation practices, this chapter report the results of a meta-review based on an analysis of a representative set of toolkit papers.

5.3.1 DATASET

To collect a representative set of HCI toolkit papers, my co-authors and I gathered 68 papers matching the following inclusion criteria.

Publication Venue and Date, Keywords: the initial selection consisted of 58 toolkit papers that were published since 2000 at the major ACM SIGCHI venues (CHI, UIST, DIS, Ubicomp, TEI, MobileHCI). We included papers containing keywords: *toolkit, design tool, prototyping tool, framework, API*. All 58 papers comply with our proposed toolkit definition.

Exemplary Papers. We then identified 10 additional papers published elsewhere, based on exemplary impact (e.g. citations, uptake) such as D3 (Bostock et. al, 2011), Piccolo/Jazz (Bederson et. al, 2004), and the Context Toolkit (Salber et. al, 1999). The total dataset includes 68 papers (**Table 5.1**). While other toolkit papers exist, our dataset serves as a representative sample from which we could (1) gather insight and (2) initiate meaningful discussion about evaluation.

VENUES

CHI	UIST
Ubicomp	DIS
MobileHCI	TEI

KEYWORDS

toolkit	framework
prototyping tool	
design tool	API

+10 EXEMPLARY PAPERS

e.g. D3, Context Toolkit

TIME



5.3.2 ANALYSIS AND RESULTS

The dataset was analyzed via several steps. I conducted open-coding (Charmaz, 2014) on a subset of our sample, describing the evaluation methods used in each publication. Next, my co-authors and I collectively identified an initial set of evaluation methods and their variations as used across papers. At this point, my co-authors and I performed focused coding (Charmaz, 2014) on the entire sample. We continued to apply the codes to the rest of the sample, iteratively refining and revisiting the coding schema. After coding all papers in our sample, we created categories (Charmaz, 2014) to derive the overarching evaluation strategies used by toolkit researchers, thus arriving at the four evaluation strategies that we identify as (1) *demonstration*, (2) *usage*, (3) *technical evaluation*, and (4) *heuristic evaluation*.

Table 5.1 summarizes the analysis, showing the count of evaluation strategies seen in the current sample. Note that this frequency count is not necessarily indicative of a strategy's overall appropriateness or success, as it only reflects the methods that researchers have applied to date from our specific sample.

#	TOOLKIT	VENUE	YEAR	REF	TYPE				#	TOOLKIT	VENUE	YEAR	REF	TYPE				#	TOOLKIT	VENUE	YEAR	REF	TYPE			
					1	2	3	4						1	2	3	4									
1	Context Toolkit	CHI	1999	[91]	■	■	■	■	24	Shared Phidgets	TEI	2007	[65]	■	■	■	■	47	PaperBox	CSC	2012	[107]	■	■	■	■
2	DENIM	CHI	2000	[60]	■	■	■	■	25	VoodooIO	TEI	2007	[101]	■	■	■	■	48	WorldKit	CHI	2012	[113]	■	■	■	■
3	Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces	CHI	2000	[62]	■	■	■	■	26	Gummy	AVI	2008	[71]	■	■	■	■	49	KinectArms	CSCW	2013	[28]	■	■	■	■
4	SATIN	UIST	2001	[41]	■	■	■	■	27	Damask	CHI	2008	[59]	■	■	■	■	50	OpenCapSense	PerCom	2013	[33]	■	■	■	■
5	Phidgets	UIST	2002	[32]	■	■	■	■	28	VoodooSketch	TEI	2008	[12]	■	■	■	■	51	ToyVision Toolkit	TEI	2013	[63]	■	■	■	■
6	Speakeasy	UIST	2003	[80]	■	■	■	■	29	GT/SD	EICS	2009	[23]	■	■	■	■	52	Sauron	UIST	2013	[93]	■	■	■	■
7	iStuff	CHI	2003	[4]	■	■	■	■	30	PyMT	ITS/ISS	2009	[35]	■	■	■	■	53	PanelRama	CHI	2014	[114]	■	■	■	■
8	MAUI Groupware Toolkit	CSCW	2004	[40]	■	■	■	■	31	Protovis	TCVG	2009	[13]	■	■	■	■	54	XDStudio	CHI	2014	[75]	■	■	■	■
9	DiamondSpin	CHI	2004	[98]	■	■	■	■	32	Sikuli	UIST	2009	[15]	■	■	■	■	55	XDKinect	EICS	2014	[74]	■	■	■	■
10	Papier-Mâché	CHI	2004	[52]	■	■	■	■	33	Intuino	DIS	2010	[103]	■	■	■	■	56	PolyChrome	ITS/ISS	2014	[3]	■	■	■	■
11	Calder	DIS	2004	[57]	■	■	■	■	34	Amarino Toolkit	MobileHCI	2010	[50]	■	■	■	■	57	PaperPulse	CHI	2015	[88]	■	■	■	■
12	ICON	ICMI	2004	[22]	■	■	■	■	35	Intelligibility Toolkit	Ubicomp	2010	[58]	■	■	■	■	58	WatchConnect	CHI	2015	[43]	■	■	■	■
13	Toolkit Design for Interactive Structured Graphics	TSE	2004	[6]	■	■	■	■	36	D-MACS	UIST	2010	[70]	■	■	■	■	59	Weave	CHI	2015	[17]	■	■	■	■
14	DART	UIST	2004	[61]	■	■	■	■	37	Prefab	CHI	2010	[20]	■	■	■	■	60	SoD Toolkit	ITS/ISS	2015	[96]	■	■	■	■
15	MaggLite Post-WIMP Toolkit	UIST	2004	[47]	■	■	■	■	38	TouchID Toolkit	ITS/ISS	2011	[67]	■	■	■	■	61	C4	TEI	2015	[51]	■	■	■	■
16	Peripheral Displays Toolkit	UIST	2004	[68]	■	■	■	■	39	D3	TCVG	2011	[14]	■	■	■	■	62	Makers' Marks	UIST	2015	[94]	■	■	■	■
17	Prefuse	CHI	2005	[38]	■	■	■	■	40	Proximity Toolkit	UIST	2011	[64]	■	■	■	■	63	Physikit	CHI	2016	[42]	■	■	■	■
18	SubArctic	CHI	2005	[46]	■	■	■	■	41	Phybots	DIS	2012	[49]	■	■	■	■	64	Retrofab	CHI	2016	[87]	■	■	■	■
19	d.Tools	UIST	2006	[37]	■	■	■	■	42	jQMultiTouch	EICS	2012	[76]	■	■	■	■	65	Let Your Body Move Toolkit	MobileHCI	2016	[84]	■	■	■	■
20	SwingStates	UIST	2006	[2]	■	■	■	■	43	PuReWidgets	EICS	2012	[15]	■	■	■	■	66	CircuitStack	UIST	2016	[104]	■	■	■	■
21	Exemplar	CHI	2007	[36]	■	■	■	■	44	.NET Gadgeeteer	Pervasive	2012	[102]	■	■	■	■	67	EagleSense	CHI	2017	[112]	■	■	■	■
22	CapToolkit	PerCom	2007	[109]	■	■	■	■	45	HapticTouch Toolkit	TEI	2012	[55]	■	■	■	■	68	Pineal	CHI	2017	[54]	■	■	■	■
23	ReactIVision	TEI	2007	[48]	■	■	■	■	46	Midas	UIST	2012	[95]	■	■	■	■									

Table 5.1. Overview of all toolkits in the sample. Types: (1) Demonstration, (2) Usage, (3) Technical Performance and (4) Heuristics.

The following sections step through the four evaluation types, summarized in **Table 5.2**. For each type, I will discuss their value and the specific techniques used. I then reflect on challenges for that type, followed by opportunities to reflect on the evaluation: opinions are based on our insights gained from data analysis, my and my co-authors' experiences and/or opinions offered by other researchers. The result is a set of techniques that researchers can use, on their own or in combination, to assess claims made about their toolkits.

5.4 TYPE 1: DEMONSTRATIONS

The now famous “*mother of all demos*” by Douglas Engelbart (1968) established how demonstrating new technology can be a powerful way of communicating, clarifying and showcasing new ideas and concepts. The transferability of an idea to neighbouring problem spaces is often shown by demonstrating application examples (Oulasvirta & Hornbaek, 2016). In our sample, 66 out of 68 papers used demonstrations of what the toolkit can do, either as the only method (19/68) or in combination with other methods (47/68). Demonstrations show *what the toolkit might support*, as well as *how users might work with it*.

This ranges from showing new concepts (e.g. Phidgets (Greenberg &

TYPE 1 - DEMONSTRATION	TYPE 2 - USAGE	TYPE 3 - PERFORMANCE
INDIVIDUAL INSTANCES <ul style="list-style-type: none"> ─ NOVEL EXAMPLES ─ REPLICATED EXAMPLES COLLECTIONS <ul style="list-style-type: none"> ─ CASE STUDIES ─ DESIGN SPACES GOING BEYOND DESCRIPTIONS <ul style="list-style-type: none"> ─ HOW TO SCENARIOS 	WAYS TO CONDUCT USAGE STUDIES <ul style="list-style-type: none"> ─ USABILITY STUDIES ─ A/B COMPARISON ─ WALKTHROUGH ─ OBSERVATION ─ TAKE-HOME STUDIES ELICITING USER FEEDBACK <ul style="list-style-type: none"> ─ LIKERT SCALES ─ INTERVIEWS 	BENCHMARK THRESHOLD BENCHMARK COMPARISON
TYPE 4 - HEURISTICS		
		CHECKLISTS DISCUSSIONS TARGETING

Table 5.2. A summary of the four evaluation strategies.

Fitchett, 2001), Context Toolkit (Salber et. al, 1999), to focused case studies (e.g. iStuff (Ballagas et. al, 2003), SoD Toolkit (Seyed et. al, 2015) to design space explorations (e.g. WatchConnect (Houben & Marquardt, 2015), the Proximity Toolkit (Marquardt et. al, 2011) and Pineal (Ledo et. al, 2017)).

5.4.1 WHY USE DEMONSTRATIONS?

The goal of a demonstration is to use examples and scenarios to clarify how the toolkit's capabilities enable the claimed applications. A demonstration is an existence proof showing that it is feasible to use and combine the toolkit's components into examples that exhibit the toolkit's purpose and design principles. These examples can illustrate different aspects of the toolkit, such as using the basic building blocks, demonstrating the workflows, or discussing the included tools. Since toolkits are a 'language' to simplify the creation of new interactive systems (Greenberg, 2007), demonstrations describe and show how toolkits enable *paths of least resistance* for authoring.

In its most basic form, a demonstration consists of examples exploring the expressiveness of the toolkit by showing a range of different applications. More systematic approaches include explorations of the *threshold*, *ceiling* or *design space* supported by the toolkit. The *threshold* is the user's ability to get started using the toolkit, while *ceiling* refers to how much can be achieved using the toolkit (Myers et. al, 2000). While demonstrations may not show the full '*height*' of the *ceiling*, they are an indicator of the toolkit's achievable complexity and potential solution space. The principles and goals of the toolkit can also

be demonstrated through a design space exploration which enumerates design possibilities (Wiberg & Stolterman, 2014) and gives examples from different points in that space.

Sometimes, a prototype toolkit may not be mature enough (e.g. due to bugs, incomplete parts) to afford other evaluation methods (e.g. a user study). In such a case, demonstrations are a way to illustrate and highlight the research concepts rather than the particular implementation. Indeed, toolkit authors should be transparent and explain that they are demonstrating concepts of the toolkit itself, and that the current version of the toolkit is not sufficiently robust for external usage.

5.4.2 EVALUATION TECHNIQUES AS USED IN DEMONSTRATIONS

Our sample reveals several techniques to demonstrate a toolkit. These techniques are not mutually exclusive and can be combined in different ways. The simplest unit of measurement for demonstration is an *individual instance*. While multiple instances can be described separately, researchers may carefully select instances as *collections* to either explore the toolkit's depth (case studies) or its generative breadth (design spaces). Toolkit authors may also *go beyond describing* the features of instances, by showing the detailed 'how to' steps involved in the instance authoring process.

5.4.3 INDIVIDUAL INSTANCES

1. Novel Examples. Demonstration of a toolkit can be done by showing the implementation of novel applications, systems or interaction techniques. The Context Toolkit (Salber et. al, 1999) is a classic case of how example applications are used to demonstrate the underlying

concepts of *context-awareness* (Schilit et. al, 1994). A more recent example is WorldKit (Xiao et. al, 2013), which demonstrates projection-based touch interfaces on everyday surfaces in four different environments. Similarly, in DiamondSpin (Shen et. al, 2004), the authors explore the capabilities of their multi-touch table toolkit by showing five different tabletop designs. Peripheral Displays Toolkit (Matthews et. al, 2004) uses three applications to demonstrate ways to enable new peripheral displays. Finally, Sauron (Savage et. al, 2013) describes three prototypes to demonstrate the toolkit's interactive features for physical prototypes. What is important is that these examples detail *how* the features, design principles, and building blocks enable new applications.

2. Replicated Examples. Toolkits often facilitate authoring of systems that were previously considered difficult to build. Recreating prior applications, systems or interaction techniques shows how the toolkit supports and encapsulates prior ideas into a broader solution space. For instance, Prefuse (Heer et. al, 2005) states that they “*reimplemented existing visualizations and crafted novel designs to test the expressiveness, effectiveness, and scalability of the toolkit*”. In d.tools (Hartmann et. al, 2006), the authors recreated a classic iPod interface, while the TouchID Toolkit (Marquardt et. al, 2011) recreated prior work from external sources (e.g. Rock and Rails (Wigdor et. al, 2011)) in bimanual interaction. Similarly, SwingStates (Appert & Beaudouin-Lafon, 2006) and Prefab (Dixon & Fogarty, 2010) illustrate the expressiveness and power of their toolkit by recreating interaction techniques in the research literature (e.g. Bubble Cursor (Grossman & Balakrishnan, 2005), CrossY (Apitz & Guimbretière,

2004)). These examples demonstrate how toolkits reduce complexity, effort and development time for recreating applications. Furthermore, replication can demonstrate how the toolkit generalizes across a variety of examples.

5.4.4 COLLECTIONS

3. Case Studies. Because toolkits often support complex applications, case studies (typically concurrent research projects) can help explore and elaborate the toolkit in greater depth. Five of our 68 papers included case studies to reveal what their toolkit can do. The iStuff toolkit (Ballagas et. al, 2003) presents case studies of other research projects that use the toolkit. Similarly, the SoD toolkit (Seyed et. al, 2015) describes its use in complex case studies: an oil and gas exploration application and an emergency response system. Prefuse (Heer et. al, 2005) reports on the design of *Vizster*, a custom visualization tool for social media data. Although case studies are less common than examples, they convincingly demonstrate the toolkit's application within large, complex scenarios as opposed to small, self-contained example applications.

4. Exploration of a Design Space. A design space exploration exemplifies the breadth of applications supported by the toolkit by fitting it into a broader research theme. Design spaces often consist of dimensions with properties (categorical or spectrum variables (Wiberg & Stolterman, 2014)) that examples can align to. A toolkit author can create a collection of examples that each examine different points in the design space. For example, WatchConnect (Houben & Marquardt, 2015) describes a design space of how the toolkit supports interaction across a watch prototype and a second screen. By providing

five examples, including both replicated and novel techniques, the authors satisfy the smartwatch + second screen design space by example. The Proximity Toolkit (Marquart et. al, 2011) similarly describes the design dimensions of proxemic interaction (Ballendat et. al, 2010) (e.g. distance, orientation, identity) and demonstrates through examples how the toolkit enables new proxemic-aware applications. Pineal (Ledo et. al, 2017) explores different ways of using and repurposing mobile sensors and outputs to author smart objects, using a combination of novel examples and replication. Finally, DART (MacIntyre et. al, 2004) is an example of a toolkit supporting the exploration of a design space through a range of ‘behaviors’ and examples. A design space exploration is thus a systematic way of trying to map out possible design boundaries. Although exploring the full design space is often impossible, examples demonstrate the breadth of designs enabled by the toolkit.

5.4.5 GOING BEYOND DESCRIPTIONS

5. ‘How To’ Scenarios. Toolkit papers can demonstrate a step-by-step breakdown of how a user creates a specific application. Scenarios break down tasks into individual steps that demonstrate the workflow, showing the results of each step. We found three ways in which toolkit authors describe scenarios. One way is to dedicate a section to describe how one example is authored (e.g. RetroFab (Ramakers et. al, 2015), Pineal (Ledo et. al, 2016)). Second, a scenario can be used throughout the paper to show how different parts of an example come together (e.g. the Proximity Toolkit (Marquardt et. al, 2011)). Demo scenarios, as in VoodooSketch (Block et. al, 2008) and Circuitstack

(Wang et. al, 2016) are common ways to explain how users might experience a toolkit's *path of least resistance*. Third, authors might include code samples. For instance, Prefuse (Heer et. al, 2004) and Weave (Chi et. al, 2015) use code snippets explaining how certain design principles or building blocks are supported directly in code.

5.4.6 CHALLENGES

Using demonstrations to 'evaluate' a toolkit poses several challenges. First is its rationale: although novel demonstrations built atop the toolkit illustrate toolkit expressiveness, it is sometimes unclear *who* would use such applications and *why*. Second, while creating demonstrations can describe '*what if*' scenarios, the demonstration itself may not show that the toolkit can indeed be used by people other than the toolkit's authors. Such lack of external validation may pose issues depending on the claims made in the paper. Third, example applications often aim to implement aspects of a potential future today; however, the target audience might not yet exist or simply be unclear. Speculating on the intended audience creates the risk of an *elastic user* (Cooper, 2004), where the definition of the target audience is stretched to accommodate implementation decisions and toolkit design. Finally, many toolkit systems, such as Marquardt et. al's Proximity Toolkit (2011), PaperPulse by Ramakers et. al (2015), and Eagesense by Wu et. al (2017), work with specialized or custom-built hardware. In creating these arrangements, the authors might alienate the potential audience, as some end-users would not be able to recreate these complicated technical setups (e.g. acquiring the appropriate equipment, creating the necessary spatial arrangements). Moreover, these specialized hardware systems might become deprecated.

5.4.7 REFLECTION AND OPPORTUNITIES

Provide Rationale for Toolkit Design and Examples. Within every piece of technology lie assumptions, principles and experiences that guide the design of that technology. Many of these assumptions can come across as arbitrary when designing toolkits. However, toolkit authors often rely on their experience even if they do not explicitly mention it. Discussing the understanding of the challenges, perhaps informed by earlier studies or experiences with other tools or toolkits, can help address why different decisions were made. Nebeling et al.'s XD toolkit suite, as used in several publications (jQMultiTouch (2012), XDBrowser (2014), XDKinect (2016)), is a compelling example of how to do this. They constructed several toolkits to structurally and systematically explore the large design space of cross-device computing. They clearly motivated the design and development of each toolkit by earlier experiences in designing toolkits and systems. More generally, research by design (Hevner et. al, 2004) helps explore concrete implementations of ideas.

First-Hand Experience. Toolkit authors often have experience creating applications that the toolkit will support, and thus are genuinely familiar with the development challenges and steps that need simplifying. This experience leads to autobiographical design (Neustaedter & Sengers, 2012) informing the toolkit design process. In Phidgets (Greenberg & Fitchett, 2001), the authors discuss their frustrations in authoring hardware-based applications, which informed their design and implementation. A toolkit may also leverage experiences with building similar toolkits. The design of D3 (Bostock et. al, 2011)

evolved from the authors' earlier experiences in creating visualization toolkits (e.g. Prefuse (Heer et. al, 2005), ProtoVis (Bostock & Heer, 2009)).

Prior Work. Challenges identified in previous research can help motivate the design of toolkits. For instance, the Context Toolkit (Salber et. al, 1999) describes challenges in authoring context-aware applications based on prior work (e.g. new types of sensing from multiple distributed sources).

Formative Studies. Authors can perform formative studies to understand their intended target audience. For instance, in d.tools (Hartmann et. al, 2006), the authors conducted interviews at product design companies. Understanding current practices can help address challenges with the design of the toolkit.

Discuss Boundaries and Underlying Assumptions. Despite including a 'limitations' section, toolkit authors often do not discuss aspects of the toolkit that do not work well. Critically discussing what does not work or the tasks complicated by the toolkit might help steer away from a 'sales pitch'.

5.5 TYPE 2: USAGE

While demonstrations answer the question of '*what can be built with the toolkit*', evaluating *usage* helps verify '*who can use the toolkit*' under certain circumstances, i.e., which tasks or activities can a target user group perform and which ones still remain challenging? To *evaluate* if and how a user group can actually use the tool, it is important to investigate how that user group uses and appropriates the toolkit. Our sample shows that more than half of the papers (35/68) include usage

studies. Only one toolkit paper uses a usage study as the only evaluation method (Houben et. al, 2016). Usage studies are often combined with demonstrations (33/68) or technical evaluations (9/68).

5.5.1 WHY EVALUATE USAGE?

The defining feature of usage evaluations is the involvement of external users working with the toolkit. Much of usage evaluation is informed by traditional user studies (as described by e.g. Dumas and Redish (1999), Lazar et. al (2017) and Nielsen (1994)), and can help verify whether the toolkit is (1) conceptually clear, (2) easy to use, or (3) valuable to the audience.

Given the prevalence of usability studies in HCI, many toolkit papers examine the toolkit's *usability* — i.e., how easy it is to use the toolkit. Common measures are users' opinions, preferences, completion time, the number of steps (e.g. lines of code), or number of mistakes. In addition, given that toolkits often propose new workflows, or enable creation of new kinds of artifacts, it is important to know if it will be useful to the target audience. In looking for *utility*, researchers inquire into the audiences' interest or outcomes. One way to assess utility is to look at the output of the toolkit. This consists of investigating the artifacts that the users authored with the toolkit. Lastly, a usage evaluation might look to understand *use* of the toolkit: how a user appropriates a toolkit, how it is used over time, and what kind of workflows are developed. The processes together with the end results can point towards *paths of least resistance*, some which may differ from the ones the toolkit authors' intended.

5.5.2 EVALUATION TECHNIQUES AS USED IN USAGE STUDIES

Given the involvement of external people in usage evaluations, toolkit authors can perform a variety of evaluations with users, each yielding different kinds of insights. Our data revealed five *ways to conduct usage studies* and two additional complementary techniques for *eliciting user feedback*. The first four techniques refer to controlled lab experiments, where participants are given consistent tasks that can yield accurate measures, such as completion time. The fifth technique is somewhat more aligned with '*in the wild*' studies, which can provide more realism as suggested by McGrath (1995) and Rogers and Marshall (2017). The last two techniques are complementary methods to elicit user feedback.

5.5.3 WAYS TO CONDUCT USAGE STUDIES

1. Usability Study. When toolkits claim that they facilitate a process, authors may choose to carry out a usability study. This can help identify issues with the toolkit, using measures of participants' performance (e.g. time, accuracy), and further qualitative feedback. Participants are typically given programming tasks that exploit various aspects of the toolkit. These programming tasks tend to be closed-ended, though some may include a small degree of open-endedness (e.g. Hartmann et. al (2007)). To increase control, some tasks may incorporate pre-written skeleton code (e.g. (Nebeling et. al, 2014)). Usability studies can examine various aspects of toolkits. For example, Papier-Mâché (Klemmer et. al, 2004) shows an evaluation of the toolkit's API usability, which revealed inconsistency in the naming of

software components and aspects of the toolkit that lacked documentation. Hartmann et al. coined the term “*first-use study*” (2006) in which participants are exposed to a toolkit for the first time and assigned different tasks. In d.tools (Hartmann et. al, 2006), the study aimed at determining the *threshold* (Myers et. al, 2000) of the system, while in Exemplar (Hartmann et. al, 2007) the aim was on determining the successes and shortcomings of the tool. The study in Exemplar (Hartmann et. al, 2007) combined close-ended tasks with a more open-ended task. Some papers report modifying the toolkit to address issues identified in a usability study, such as Papier-Mâché (Klemmer et. al, 2004) and DENIM (Lin et. al, 2000), which Greenberg and Buxton (2008) suggest should be the main goal of usability studies.

2. A/B Comparisons. One way to suggest improvement over existing work is to compare the new toolkit to a baseline. Baselines include not having a toolkit, or working with a different toolkit. In MAUI (Hill & Gutwin, 2004), the authors compare different platforms to measure what they defined as *effort*: number of classes, total lines of code, lines written for feedthrough and development time. By comparing it to GroupKit (a prior toolkit that supports a similar task (Roseman & Greenberg, 1996)), and Java (no toolkit), the authors can show the degree of improvement from the current state-of-the-art. A/B comparisons could test for variations within the toolkit. Lin and Landay in Damask (2008) compared a full version of their prototyping tool to one without the key features (patterns and layers) to determine the improvement and preference. Finally, both Paperbox (Wiethoff et. al, 2013) and XDStudio (Nebeling et. al, 2014) compare different configurations of their toolkit.

3. Walkthrough Demonstrations. A walkthrough demonstration consists of showing the toolkit to a potential user and gathering their overall impressions. Unlike *cognitive walkthroughs* (Polson et. al, 1992), walkthrough demonstrations are not about the user working directly with the tool to identify usability problems. In a walkthrough demonstration, the experimenter has full control and explains the workflow to participants, together with examples and even limitations. This approach is particularly suitable when toolkit creators want to get feedback on the utility of their toolkit, as it removes the focus from using the toolkit (as one might find in a usability study) and shifts it towards the value of having the toolkit. While the walkthrough technique has not been explored extensively, RetroFab (Ramakers et. al, 2016) is an example of this approach. This technique can be useful to gather feedback on the idea rather than the specific toolkit implementation, and might serve for toolkits that are not ready for usability testing or deployment.

4. Observation. Direct observation helps inform how users approached the toolkit to solve problems ranging from closed tasks requiring a specific solution to a given problem, to open tasks where participants formulate the problem and use the toolkit to create their own solution. While our analyzed papers rarely presented any in-depth discussion of participants' processes or workflows, they did provide examples of the toolkit's use. HapticTouch (Ledo et. al, 2012) tested participants' ability to transfer concepts about haptics, which were provided at varying levels of abstraction, into an interactive application: its authors assessed the *paths of least resistance* the toolkit afforded to solve both open and close-ended tasks. Our analysis also saw observational studies used within short-term (Let Your

Body Move Toolkit (Pfeiffer et al, 2016)) and long-term (C4 (Kirton et. al, 2016), and Intuino (Wakita & Anezaki, 2010)) workshop settings involving multiple participants. For example, Pfeiffer et al. (2016) asked participants to brainstorm ideas and create Wizard-of-Oz prototypes using the toolkit. Their video analysis discusses the applications created, as well as in-depth details of how their creations were made. In C4 Kirton et. al (2013), participants attended 3-week workshops, with some staying further for a 4-week artist residency: observation informed its creators on how design decisions held up in the implementation.

5. Take-Home Studies. Some external validity (McGrath, 1995) can be acquired by conducting experiments outside lab settings. While it is difficult to deploy a toolkit before it has gained broader acceptance, researchers can provide their toolkit to “early adopter” participants. Participants receive the toolkit (and all necessary components and documentation) to create any applications of their liking within a given timeframe (e.g. a week). Phidgets (Greenberg & Fitchett, 2001), jQMultiTouch (Nebeling et. al, 2012) and the Proximity Toolkit (Marquardt et. al, 2011) are iconic examples where students in an advanced HCI class were given access to the toolkits and necessary hardware components to create interesting examples as a prompt. They all demonstrate how students could easily work with the proposed constructs, where they focused on design aspects of the assignment versus low-level coding.

5.5.4 WAYS TO ELICIT USER FEEDBACK

6. Likert Scale Questionnaires. Likert scales provide a non-parametric value pertaining to a question. The questions can later be analyzed

either through non-parametric tests or by examining the median values. In toolkit research, while often acting as validation of claims (e.g. ease of use), Likert scales can formalize the results to clarify a hypothesis. For instance, in Exemplar (Hartmann et. al, 2007), the authors were unsure as to whether the system empowered both experts and non-experts, as the performance between these two can differ considerably. By using Likert scale questionnaires, participant responses confirmed that both experts and non-experts felt empowered, thus validating their hypothesis. Other examples like Damask (Lin & Landay, 2008), d.tools (Hartmann et. al, 2006), Paperbox (Wiethoff et. al, 2013) and Panelrama (Yang & Wigdor, 2014) use Likert scales to quantify user feedback on their system. This feedback often complements other usability results.

7. Open-Ended Interviews. In our sample, 12 papers ask participants about their experiences or challenges performing their tasks, which provided the authors with insight in terms of processes, successes and shortcomings of the toolkit (e.g. Prefuse (Heer et. al, 2005), Physikit (Houben et. al, 2015) and PanelRama (Yang & Wigdor, 2014)). Interview questions can start from a script, but are open in that they allow further inquiry as opportunities arise, such as pursuing interesting and/or unclear responses. Quoting participants gives life and adds strength to findings (e.g. Weave (Chi & Yang, 2015), DENIM (Lin et. al, 2000), Midas (Savage et. al, 2013)). Interviews can also expose how users perceive toolkit features, and can contextualize other usage data.

5.5.5 CHALLENGES

Evaluating the toolkit's implementation through usability tests could distract from the conceptual ideas as well as the opportunities facilitated by the toolkit. Olsen (2007) warns against falling into “*the usability trap*”, as the three underlying assumptions for usability evaluation – walk up and use, standardized tasks, and problem scalability – are rarely met for systems research. Additionally, toolkits in HCI research are still prototypes. It is difficult for a small team to create a toolkit with the quality of a commercial product (*fatal flaw fallacy* (Olsen, 2007)). Controlled experiments measuring usability are limited in scope and evaluate a very small subset of what the toolkit can accomplish, making it difficult to generalize usage results. Furthermore, selected experimental tasks might favour elements that the toolkit can accomplish. In achieving control of the tasks, researchers may optimize for these tasks, or only create what a usability test can measure (Olsen, 2007).

While observations of people using the toolkit provide information about use, they may not assess how the toolkit fares in the real world. McGrath (1995) discusses this as the trade-off between realism, precision and control. Even in “take home” studies, realism is compromised: participants are given all necessary components, instruction, access to resources (e.g. documentation, direct access to the toolkit creators). This creates an idealistic scenario not necessarily present in real-world adoption (Ledo et. al, 2017). Furthermore, it is difficult to identify appropriate participants for usage evaluations, especially as toolkits propose new ways to solve a problem. Specialized target

audiences may not even exist yet (Nebeling, 2017). Given the academic context, it is often easiest to find student populations. Students (e.g. computer science students) are often used as a stand-in for the target audience (e.g. developers), assuming that if students can use the toolkit then professionals might too. However, results may not always transfer to the intended target audience. Toolkits often require extensive use before becoming familiar. Thus, a premature evaluation can set up the toolkit for an unfair comparison.

Another challenge in evaluating usage is that a toolkit system may facilitate unfamiliar ways to solve a problem, which may fundamentally change how an end-user programs or solves a particular problem. For example, if a toolkit primarily provides an API that wraps certain functionality, a proficient programmer might quickly be able to adopt it and work with it. In contrast, a programming by example approach would change how a programmer solves the problem: instead of writing code the programmer has to manipulate virtual objects visually, and define positive and negative examples. Moreover, end-users may have months or years of training in their current tools and ecosystems, which may shape their existing practices and expectations, thus affecting how they receive the new toolkit. In this type of cases, a usage study is more about understanding whether end-users can quickly adopt an alternative way of thinking, or to examine whether they can create new algorithms given the new toolkit's approach.

One last challenge of usage studies is the conceptual threshold. Toolkit systems sometimes encapsulate research concepts (e.g. Proxemic Interaction) which may not be familiar to potential study participants. As a result, it may not be possible to discern whether issues or

limitations of a participant's prototype are due to technical gaps or conceptual gaps (i.e. participants do not create a particular type of scenario because they do not understand the concept).

5.5.6 REFLECTION AND OPPORTUNITIES

Bringing Utility into the Picture. A central challenge of usability evaluation is its focus on toolkit usability *vs.* utility (Greenberg & Buxton, 2008): while a toolkit may be usable, it may not be useful. Similarly, a toolkit may have sufficient utility to make users satisfied in spite of usability issues being present. Walkthroughs and interviews can help here, where questions about utility can be raised and responses explored in depth.

Selecting Tasks and Measures Carefully. While more control, more measures and more quantifiable results seemingly provide rigour, we argue that rigour is only of value if truly representative tasks and appropriate measures are used. Rigour should come from a careful selection of the method, technique, and means of executing the technique. Publications should clearly articulate why the chosen tasks and measures support the claims made in the paper (Greenberg & Buxton, 2008).

Recognizing the Consequences of Audience Choice. Toolkit authors should critically reflect and understand the implications of their choice of audience to study. As mentioned, the audience can be a close approximation or a starting point, but authors need to articulate such implications and limitations.

5.6 TYPE 3: TECHNICAL PERFORMANCE

While demonstrations and usage studies evaluate *what* a toolkit can do and *who* might use that toolkit, researchers can evaluate the technical performance of the toolkit to find out *how well* it works. From our sample of 68 toolkit papers, about one third of the papers (18/68) include technical performance studies. Technical studies are complementary to demonstration and usage evaluations, as they convey additional information on the technical capabilities of the toolkit.

5.6.1 WHY ANALYZE THE TECHNICAL PERFORMANCE?

The goal of studying technical performance is to benchmark, quantify or analyze the toolkit or its components to verify or validate the performance. Technical performance can be measured in terms of efficiency (e.g. speed of the algorithm, throughput of a network protocol), precision (e.g. accuracy of an algorithm, fault tolerance), or comparison against prior techniques. Overall, the purpose is, thus, to measure some form of system performance. These measures show whether it meets basic usage standards (*threshold*), or if there are improvements over the state-of-the-art. Technical benchmarks can push the boundaries of the toolkit to show when it no longer works as expected. Authors sometimes turn to software engineering metrics (e.g. lines of code, number of classes) to show improvement over existing practices.

5.6.2 TECHNIQUES AS USED IN TECHNICAL PERFORMANCE

The Software Engineering community has a rich set of tools to evaluate the performance of systems (Blackburn et. al, 2006). Our dataset

showed that toolkit authors examine a wide variety of benchmarks such as: *website loading time* (D3 by Bostock et. al (2011)), *spatial resolution* (OpenCapSense by Grosse-Puppendahl et. al (2013)), *frame-rate* (KinectArms by Genest et. al (2013) and C4 by Kirton et. al (2013)), *GPU usage* (C4 by Kirton et. al (2013)), *memory allocation* (e.g. Protopvis by Bostock and Heer (2009) and C4 by Kirton et. al (2013)), *load time* (Protopvis by Bostock and Heer (2009)), *lines of source code* (Swingstates by Appert and Beaudouin-Lafon (2006) and Context Toolkit by Salber et. al (1999)), *size of binary* (Swingstates by Appert and Beaudouin-Lafon (2006)). Performance metrics should be tied to the claims of the paper, and the needs that must be satisfied for the toolkit to be operational or go beyond the state-of-the-art.

1. Benchmarking Against Thresholds. For certain types of applications, systems and algorithms, there are known, tested or desirable thresholds that serve as baseline to verify that a system meets a commonly accepted standard of use (e.g. accuracy, latency). For instance, 30 fps is often used for real-time tracking systems (Newcombe et. al, 2011). Both KinectArms (Genest et. al, 2013) and EagleSense (Wu et. al, 2017) present new tracking systems benchmarked at this 30 fps rate. Thresholds can be derived empirically, technically or from experience using the tools.

2. Benchmarking Against State-of-the-Art. Benchmarking often looks for improvements over existing state-of-the-art solutions. This comparison approach is often similar to algorithm contributions in HCI, such as the \$1 Gesture Recognizer (Wobbrock & Wilson, 2007), where a toolkit's capabilities are compared against well-known baselines, or the best algorithm for that purpose. For instance,

in OpenCapSense (Grosse-Puppendahl et. al, 2013), the authors compared the toolkit's capacitive sensing performance to the earlier CapToolKit (Wimmer et. al, 2007). While not a toolkit (and thus not part of our dataset), the \$1 Gesture Recognizer (Wobbrock & Wilson, 2007) is an excellent example of benchmarking against the state-of-the-art: the benchmarks showed that it was considerably close to the state-of-the-art, yet much simpler to implement (about 100 lines of code). D3 (Bostock et. al, 2011) compared page load time to a prior toolkit and to Adobe Flash. Page load time was deemed important given their use-case: rendering visualizations created with the toolkit on the web.

5.6.3 CHALLENGES

Technical benchmarks often complement demonstrations or usage studies. Measuring technical benchmarks in isolation may highlight some human aspects of using a toolkit (e.g. frame rate, latency), but do not account for what it is like to use the toolkit. For instance, representative examples may still be difficult to program, even if requiring few lines of code. Similarly, a paper may not always (explicitly) clarify the benchmark's importance (e.g. 30 fps in EagleSense (Wu et. al, 2017)). Another challenge is that benchmark testing relies on comparisons to an existing baseline. If performance specifications have not already been published, authors must access state-of-the-art systems to perform the comparisons. Given the prototypical nature of HCI toolkits and the *fast-moving targets* of technology (Myers et. al, 2000), many pre-existing baselines may already be deprecated or require extensive reimplementation by the toolkit authors. Alternatively, a baseline may not exist, as the technical challenge may not

have been solved before (Olsen, 2007). Research HCI toolkit developed by a single person or a small team may be far from optimized. Thus, sub-optimal performance does not signify how the system could fare. For instance, early versions of GroupKit (Roseman & Greenberg, 1996) had slow network performance and suffered from multiple race conditions, as the authors were placing their efforts elsewhere. Yet, these bottlenecks were not fundamental to the toolkit – many research concepts were still explored and realized working with GroupKit, such as GroupWeb (Greenberg & Roseman, 1996), and TeamRooms (Roseman & Greenberg, 1996b).

5.6.4 REFLECTION AND OPPORTUNITIES

Contextualize and State Technical Limitations. HCI toolkit researchers often have quite different goals from commercial toolkit developers. For example, researchers may want to show how interaction concepts can be packaged within an easy-to-program toolkit (e.g. its API), where the underlying – and perhaps quite limited – infrastructure only serves as proof of concept. Significant limitations should be stated and contextualized to explain why they do not (or do) matter.

Risky Hypothesis Testing. Toolkit authors should openly discuss the rationale behind the tests performed and whether the tests are a form of stress testing. Similar to some of Greenberg and Buxton's arguments (2008), perhaps the best approach is to actively attempt to break the toolkit's proposed technical claims (e.g. EagleSense's ability to accurately track up to four people in real-time (Wu et. al, 2017)) to truly understand the toolkit's technical boundaries. One way to test these boundaries is to stress-test the system's scalability for a chosen metric.

Open Source and Open Access. As toolkit researchers, we can facilitate comparison and replication by making our work available to help future researchers, as done by toolkits such as D3 (Bostock et. al, 2011), the Proximity Toolkit (Marquardt et. al, 2011), Midas (Savage et. al, 2012). Ideally, this goes beyond the academic publication or the toolkit source code and documentation, but also includes the benchmarking data so that others can run the tests (e.g. on different computers or as baselines for future studies).

Discuss Implicit Baselines. While a toolkit paper may assume standard metrics to determine that a system works (e.g. 24 fps, or few lines of code to accomplish a task), it may help to mention why this metric is relevant. Thus, less familiar readers can better understand the performance implications.

5.7 TYPE 4: HEURISTICS

Heuristics in HCI are typically associated with Nielsen et al.'s (e.g. Molich and Nielsen (1990), Nielsen (1993)) discount method to informally assess interface usability. Given the challenges of toolkit evaluation, toolkit researchers have devised toolkit-centric heuristics (guidelines) to assess the end-result of a toolkit, such as Blackwell et. al's Cognitive Dimensions of Notations (2000) and Olsen's discussion on evaluating interactive systems research (2007). The toolkit is then inspected against these heuristics, which in turn serves to inform strengths, weaknesses, and reflection of the toolkit's potential value. The heuristics have been extracted from tried and accepted approaches to toolkit design and have been used by others. In the dataset. Blackwell et. al's heuristics (2000) are used in Protovis (Bos-

tock et. al, 2009) and Exemplar (Hartmann et. al, 2007), whereas Olsen's heuristics are used in WatchConnect (Houben & Marquardt, 2015), Intelligibility Toolkit (Lim & Dey, 2010), D-Macs (Meskens, et. al, 2010), Gummy (Meskens et. al, 2008), XDKinect (Nebeling et. al, 2014), and Society of Devices Toolkit (Seyed et. al, 2015). In our sample, heuristics always complemented other methods.

5.7.1 WHY USE HEURISTICS?

Heuristics are used as a discount method that does not require human participants to gather insight, while still exposing aspects of utility. Olsen's ideas of *expressive leverage* and *expressive match* (Olsen, 2007) resonate with Greenberg's view of toolkits as a language that facilitates creation (2007), or Myers' themes of successful systems *helping where needed* and creating *paths of least resistance* (2000). Heuristics are based on tried success (Olsen, 2007) or theories (e.g. cognitive dimensions (Blackwell et. al, 2000)).

Blackwell et. al's (2000) Cognitive Dimensions of Notation (CDN) was initially offered as a set of discussion points that designers could also use as heuristics to verify system usability. Their primary goal was to create a vocabulary for experts to make early judgements when designing, and to articulate decisions later. The authors describe it as a synthesis of several sources that can partially address elements of the interface design process. CDN also included a questionnaire approach (Blackwell & Green, 2001) to guide and structure user feedback sessions.

Olsen's heuristics (2007) aimed to bring the focus of toolkit evaluation back to what he saw as the value of UI systems research, which

corresponds to our aforementioned reasons why HCI researchers build toolkits. Olsen provided terminology and means to support common claims made in toolkit papers. Interestingly, Olsen states that given a set of claims, one can *demonstrate* how the toolkit supports them, which may explain why our data shows prevalent combinations of Type 4 evaluations together with Type 1 (demonstrations).

Following a comprehensive list of heuristics can help identify areas not addressed by the toolkit. Some heuristics might be more crucial (e.g. *problem not previously solved* (Olsen,2007)). Conversely, some may not be relevant for the proposed toolkit (e.g. *secondary notations* (Blackwell et. al, 2000)). Heuristics can and should be omitted when appropriate (Molich & Nielsen, 1990).

5.7.2 EVALUATION TECHNIQUES FOR HEURISTICS

I identified three ways to carry out a heuristic evaluation: checklists, discussion, and as a basis for usage studies.

1. Heuristic Checklists. The checklist approach consists of selecting a heuristic evaluation approach and going through individual heuristics one at a time. In doing so, authors can reflect on whether the toolkit satisfies the heuristic or not, and the extent of meeting it. For instance, Hartmann et al. (2007) followed Blackwell and Green's CDN through a questionnaire (2001). In evaluating each item, they found that many the limitations of the system were due to the inability to show many sensor visualizations at once. Similarly, Meskens et al. (2010) follow Olsen's heuristics to determine which elements of the interface are lacking (e.g. ability to generalize and reuse).

2. Discussion / Reflections based on Heuristics. In contrast to the checklist approach, Olsen's heuristics (2007) are also used as reflection points in the discussion of a toolkit paper. This reflection allows the authors to better understand the limitations and whether there are issues in the toolkit that are not addressed. Both Gummy (Meskens et. al, 2008) and WatchConnect (Houben & Marquardt, 2015) are examples of this approach, where authors reflect on shortcomings (and ways to address them) as well as compare their toolkits to the state of the art.

3. Basing Usage Studies on Heuristics. Heuristics can help determine what is useful to evaluate. XDKinect (Nebeling et. al, 2014) tailored their usage study to some of Olsen's guidelines (2007), such as reducing solution viscosity and ease of combination.

5.7.3 CHALLENGES

A danger of heuristic evaluations is falling into self-fulfilling prophecies, where authors stretch definitions of the heuristics to justify their claims. Alternatively, authors might choose to only focus on (1) heuristics that their toolkit addresses or (2) how the toolkit addresses them without acknowledging the negative aspects or compromises (e.g. increasing *flexibility* at the expense of *expressive match*). Sometimes the heuristics are not relevant to a particular toolkit. For example, CDN (Blackwell et. al, 2000) covers a breadth of applications, where some heuristics only apply to one group (e.g. visual programming environments). Omitting heuristics without clear rationale could lead readers to believe that the authors are cherry picking heuristics. Heuristic evaluations are often carried out by the authors, who may have an implicit bias. While heuristic evaluation in HCI suggests

that external evaluators add value to the process ((Molich & Nielsen, 1990),(Nielsen, 1993)), it proves difficult for toolkits given that external evaluators would need to be competent in the toolkit domain and understand the associated design and research concepts. None of the surveyed papers used external evaluators.

5.7.4 REFLECTION AND OPPORTUNITIES

Using Heuristics as Design Guidelines. Heuristics can serve complementary purposes: they can inform design as well as help evaluate designs. Thus, toolkit authors can conceptually consider *how* to support aspects of creation early on through best practices (e.g. API practices (Stylos et. al, 2008)). As examples, the Intelligibility Toolkit (Lim & Dey, 2010) and HapticTouch (Ledo et. al, 2012) both discuss heuristics inspiring some of their design goals.

Using Heuristics to Inform Techniques from Prior Types. Given the vocabulary provided by heuristics, authors can consider how demonstrations or usage studies might stem from the heuristics themselves. For example, toolkit authors could choose to evaluate the *expressive match* (Olsen, 2007) within one part of the toolkit, which could be executed through a usage study (e.g. A/B comparison, observation). Olsen (2007) suggests that one way researchers might evaluate *expressive match* is to perform a “design flaw test”, where participants are asked to remedy a flaw using a design with “good expressive match” and a baseline deficient design with “bad expressive match”. As an example, Olsen uses colour selection within a drawing application, where a researcher might compare using a mouse-based colour picker to manually entering hexadecimal code values into a textbox.

Transparency. Toolkit authors can disambiguate cherry picking versus ignoring irrelevant heuristics by articulating why a heuristic is or is not considered. This will increase transparency and possibly expose gaps in the evaluation.

5.8 DISCUSSION

The meta-review in this chapter reveals 4 strategies to *evaluate* toolkits: (1) *demonstrations* (what a toolkit can do), (2) *usage* (who can use the toolkit and how), (3) *technical evaluations* (how well a toolkit performs), and (4) *heuristics* (to what extent the toolkit meets standard guidelines).

My co-authors and I wanted to get a more detailed sense of the data, which prompted us to re-code the papers looking and interpreting the types of contribution. We performed two passes to the data to ensure consistency. **Figure 5.3** summarizes the entirety of the data analyzed, including the distribution of the individual evaluation techniques. The next section offers several opinions, formed from: the building experiences of myself and my collaborators; the meta-review analysis; and from the writing of other toolkit researchers.

5.8.1 RETHINKING EVALUATION

Rather than considering some methods as *better* than others, we believe that it is more important to use methods that best match the claims of the toolkit paper, and what that evaluation method might yield. One way to determine this might be for authors to ask themselves: *if the evaluation technique were to be removed, what is the impact to the paper?* In answering that question, authors might realize the essential methods, and which ones are secondary or even unnecessary.

Evaluation by Demonstration?

One central observation in our review is that demonstrations are by far the most common way to communicate the functionality of the toolkit. Demonstrations vary in complexity, ranging from small examples to complex interaction techniques and systems. 19 toolkit papers used demonstration as the only way to communicate or evaluate the toolkit's capabilities. Novel and replicated examples are quite common due to their easy implementation and description. However, further analysis showed that it is rare to find more systematic explorations of the capabilities of toolkits through case studies concurrent to the time of publication, or design space explorations. Moreover, many toolkit papers combine examples with code snippets and how-to scenarios to help the reader understand *what* the toolkit supports. While demonstrations are often not considered a formal *evaluation*, they show evidence through "research by design" (Hevner et. al, 2004) and are highly effective in communicating the principles, concepts and underlying ideas of the toolkit. In fact, using the toolkit to create prototypes can lead to refinements in the toolkit itself, as was done in SATIN (Hong & Landay, 2000). When linked back to the five goals of toolkit research, demonstrations provide the most complete and compelling evidence for achieving the goals of designing the toolkit. The wide adoption of *evaluation by demonstration* indicates that such well explored examples can be a measure of success for the underlying concepts and ideas of a specific toolkit implementation.

Usability Studies (Still) Considered Harmful Some of the Time

Half of all toolkit papers in our sample conducted usage studies. These include compelling examples examining how people work with

a toolkit; how a toolkit is used and appropriated in a realistic environment; or how toolkits enable creativity and exploration. Although usage studies play a fundamental role in establishing *who* can use a toolkit, our analysis shows that many authors still fall into the ‘*usability trap*’ (Olsen, 2007). Despite Greenberg and Buxton’s warning that usability studies can be ‘*harmful*’ if not applied to the right problem (2008), many papers in our sample performed usability studies to evaluate complex toolkits. Such studies may employ artificial tasks, small sample sizes, and non-representative user groups to evaluate a small subset of solution paths offered by the toolkit. While still yielding results, these are limited to that specific task, and rarely generalize to the entire toolkit capabilities, development paths, broader audience that would use the toolkit, and the context around toolkit learning and use.

Echoing prior work discussing that usability studies are not always required for toolkit research including Hudson & Mankoff (2014) as well as Olsen (2007), we believe narrow usability studies as currently done by most toolkit authors at best play only a *minor* role establishing or evaluating the novelty or significance of the toolkit and its underlying ideas. If done narrowly, they should at least be combined with other techniques: all but one paper in our sample also included demonstrations or technical evaluations. Even so, we consider this a widespread application of a weak mixed method approach, where researchers may make – perhaps unwarranted – generalized usability claims across the entire toolkit. Careless usability evaluations can be costly, as they may evaluate the wrong possible futures and lead to

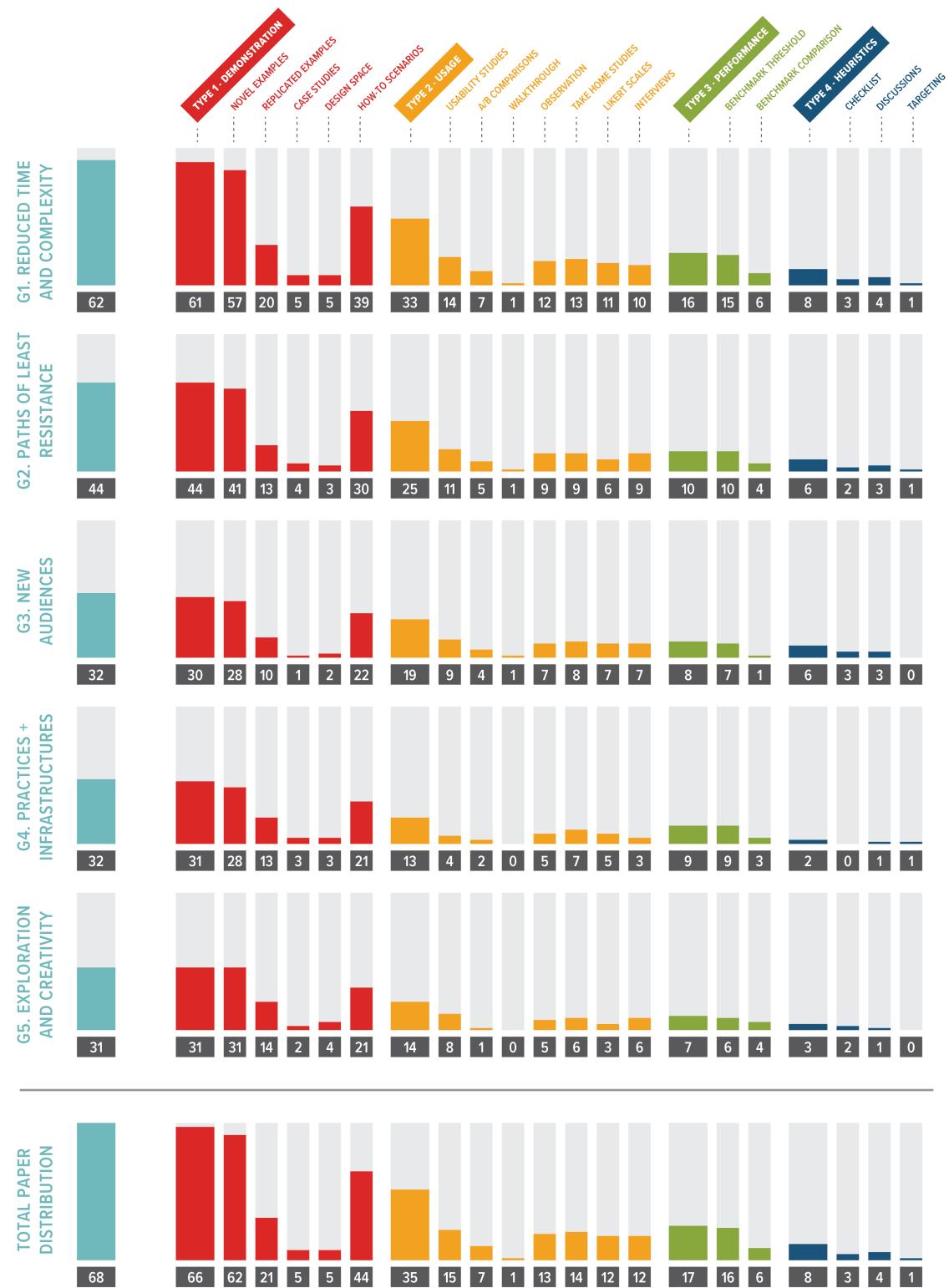


Figure 5.3. Summary of the entire dataset for toolkit evaluation. Each row represents the different toolkit goals and their (interpreted) method distributions, while the last row shows the entire data distribution.

false conclusions (Salovaara et. al, 2017). Usability studies can evaluate some parts of the toolkit, but they must be designed and conducted with care.

Successful Evaluation versus Successful Toolkit

In our dataset, we observed a diverse range of toolkits that address various sub-fields within the HCI community, where there is no indication that the success of the toolkit was necessarily tied to the success of the evaluation. Some of these toolkits have had enormous impact within the research community. For example, the Context Toolkit (Salber et. al, 1999) has had a transformative effect on research within the space of context awareness, as evident from the 1326 citations. Other toolkits have moved on to become successful outside of the research community. For instance, D3 (Bostock et. al, 2011) has been widely adopted for web-based interactive visualizations. Their paper already suggested that the evaluation may not be indicative of success: “*while we can quantify performance, accessibility is far more difficult to measure. The true test of D3’s design will be in user adoption*” (Bostock et. al, 2011). Success can also lie in enabling new research agendas. The Proximity Toolkit (Marquardt et. al, 2011) operationalized proxemic interaction concepts into concrete building blocks and techniques. Many downloaded the toolkit for research or to learn how to build proxemic-aware applications.

Non-Coding Toolkits

It is possible to filter out the dataset to only account for toolkits in which end-users do not have to write code. Through visual inspection of this very limited data-set (**Figure 5.4**), there are 2 speculations that

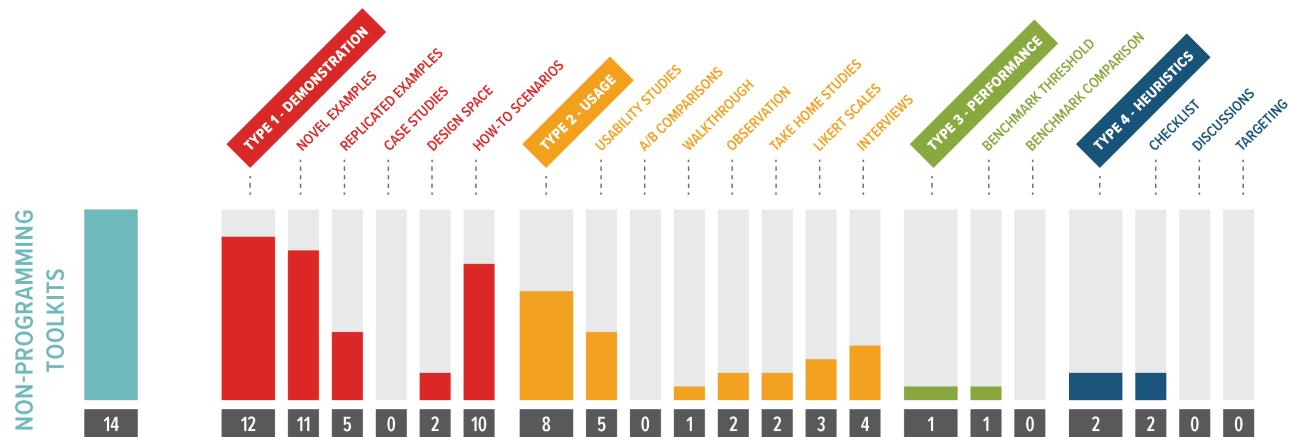


Figure 5.4. Distribution of evaluation strategies for non-coding toolkits. Highest value on the chart is 14 papers.

can be made. First, how-to-scenarios (10/14 papers) might be an effective strategy to communicate the tool to readers. Non-programming toolkits often provide a user interface that guides the end-user towards generating different solutions. In a user interface, the *paths of least resistance* are more clearly delineated than in code, since a visual interface can force specific ordering of operations. In that sense, the tools can more explicitly limit the flexibility in favour of a higher expressive match with more potentially predictable outcomes. Second, there appears to be a high ratio of usage evaluations (8/14). There might be two possible explanations to this high ratio. Non-programming toolkits typically have an explicit audience (e.g. interaction designers), which might instigate reviewers' expectations for usage studies. On the other hand, non-programming toolkits typically leverage known technical details into a new functionality and workflow. Given that the research community tends to favour technical novelty (Fogarty, 2017), non-programming toolkits likely have no choice but to include usage studies to satisfy expectations in peer-review. Prototyping tools have idiosyncrasies that make evaluation particularly tricky, as discussed below.

Meta-Development

Unlike a traditional system implementation, a prototyping tool needs to create more than one possible interactive solution. This means that a researcher cannot simply go from a sketch to a prototype. Instead, the researcher must build a general-enough approach that can generate a desired prototype. By general-enough, I mean that the tool has to generate the desired prototype while (1) accommodating for alternate versions, and (2) integrating the authoring process of that prototype into the desired *path of least resistance*. If the researcher incrementally updates the tool to increase the vocabulary and the number of possible outcomes, the entire tool needs to evolve as well. Alternatively, the researcher would need to know all the building blocks ahead of time, and then build the tool. The second approach is more difficult, given that the understanding of the problem space, the vocabulary, and the target prototypes are often part of a reflection and evolution that takes place while the tool is being built.

Existing Scaffolding

When creating a traditional programming toolkit, researchers can rely on an existing platform (in this case, the programming language) to act as the existing scaffolding for future end-users. This means that if, for instance, a toolkit is built extending JavaScript, (1) the rules and syntax of the language are pre-existing and well-defined, (2) re-ordering of operations would lead to an error, and (3) there is instant power in combination (Olsen, 2007), as other web-based toolkits can be integrated. In contrast, a tool requires (1) creating specific rules from scratch, (2) ordering of operations should be explicitly defined and may or may not immediately scale, and (3) power in combination is

constrained to how the tool can leverage external resources beyond the application itself.

One way of enabling this power in combination is by having the tool only address a portion of the process. For instance, Sauron (Savage et. al, 2013) converts inputs captured by a camera into a range of explicit values which are sent as socket data to external applications. As a result, custom applications (or applications using a similar protocol) can retrieve this data to create interactive prototypes, often requiring end-users to write code. Alternatively, the tool can explicitly use other applications to provide additional functionality. For example, Retrofab (Ramakers et. al, 2016) was programmed using the Mesh-Mixer API to communicate with MeshMixer (an external 3d modelling application) so that the tool could appropriate 3d modelling functionality. Tools can also leverage the operating system, as done by PreFab (Dixon & Fogarty, 2010), where the system reverse-engineers the operating system screen to overlay new outputs, and provide new input transformations by manipulating the operating system's mouse events. Lastly, a tool may simply be self-contained, meaning that all of the functionality resides within that application. An example includes Sketch-n-Sketch (Hempel & Chugh, 2016), which allows programmatic generation of SVGs. While Sketch-n-Sketch relies on external libraries, its functionality remains self-contained to that application.

When considering these strategies to achieve power in combination, it becomes clear that there are additional challenges for evaluation. Interface elements need to all support the paths of least resistance, and the design and testing of the interface must account for many

combinations of user input, which may be harder to fix given the meta-development approach. If the prototyping tool leverages an external program, it has to adapt the external tool's path of least resistance, to the one of the prototyping tool. This might limit the possible user actions, or the types of actions that can truly be tested. Lastly, if a tool requires designers to write code to generate outputs (e.g. as done in systems that send encoded network messages), then there are less claims that can be made about the building blocks and ways of thinking afforded by that tool.

End-User Practice

While it is true that many times systems are designed to support a particular practice, it is also the case that tools shape what is possible and provides a way of thinking in for that practice, a thought echoed by Greenberg (2007) and Myers et. al (2000). This means that over time, practices can become exclusively about what the current tools can support. As a result, a new tool will often conflict with the existing approach, either because that element of the practice is not common at the time, or because it offers an alternative, unfamiliar way to solve a problem. Comparisons, in that case, would be unfair, as end-users have had extensive time learning existing tools, compared to an immediate exposure to a new prototyping tool.

The Need for HCI Infrastructure Research

This chapter argues that toolkits have profoundly influenced HCI research and will continue to do so in the future. Going back to the pioneering work of Engelbart (1968), Sutherland (1980), or Weiser (1991), we observe how invention through building interactive sys-

tems, architectures and frameworks enabled early researchers to explore completely new spaces. Since then, there has been an enormous growth in toolkits exploring technical realizations of concepts, techniques and systems in many emerging areas within the field (e.g. physical computing, tangible interfaces, augmented reality, ubicomp) and demonstrating new possible futures.

HCI systems and toolkit research serves to further develop and realize high-level interaction concepts (e.g. proxemic interactions (Marquardt et. al, 2011)). Consequently, toolkits make these conceptual ideas very concrete, and enable further conversations and follow-up research. For instance, the Context Toolkit (Salber et. al, 2000) was a very successful toolkit that moved research in context-aware computing (Want et. al, 1994) forward by enabling developers to rapidly prototype context-aware applications. The toolkit provided a component-based architecture separating context inference from the applications that used context information and allowing developers to respond to context changes in an event-driven way. By making these ideas (and their realization in software) very concrete, the Context Toolkit also fueled criticism from researchers who argued that a computational representation of context, as encapsulated in the toolkit, did not capture the complexity of how people behave in the real world. Greenberg (2001) argued that many contextual situations are not stable, discernable, or predictable, and argued for context-aware applications to explain the inferred context and how they respond to it (what Bellotti & Edwards refer to as “intelligibility” (2001)). Interestingly, these discussions led to development and integration of these ideas in future systems and toolkits, such as the Situations

Framework (Dey & Newberger, 2009) and the Intelligibility Toolkit (Lim & Dey, 2010).

5.9 LIMITATIONS

My co-authors and I make no pretense that our overview of evaluation strategies for toolkits is complete. First, to ensure that our meta-review focused on forms of evaluation that are relevant to currently accepted standards, we limited our sample to recently published toolkit papers. Thus, we may have missed forms of evaluation used in past toolkit research. Second, many research projects make multiple contributions not captured in a single paper. Our analysis only reflects what is described in that single paper. For some of the toolkits in our meta-review, additional evaluations were described in later publications (e.g. Prefab (Dixon & Fogarty, 2010)). Finally, my co-authors and I have all built and designed toolkits. While our reflection of toolkit evaluation strategies is likely strengthened by our first-hand experience, it may also have introduced bias.

5.10 CONCLUSION

Research toolkits have fundamentally influenced and shaped the way interactive technology is built, and will continue to do so. Despite the impact and success of toolkits, evaluating them remains a challenge.

In this thesis, I consider the strengths and weaknesses of the various evaluation methods as discussed above when deciding how to evaluate the prototyping tools discussed in subsequent chapters. In particular, I considered the claims I make with each prototype, and whether a particular evaluation method would help substantiate those claims.

PART 2

SOUL-BODY

PROTOTYPING

CHAPTER 6. SOUL–BODY PROTOTYPING

This thesis proposes *repurposing existing hardware (i.e. mobile phones and watches) and software to enable designers to create live interactive prototypes for smart interactive objects without coding or create custom circuitry*. This chapter¹ examines a way to circumvent the need for electronics and programming via what I refer to as the Soul–Body Prototyping Paradigm. This paradigm is a metaphor that suggests placing mobile devices inside of fabricated enclosures to create new smart object prototypes. The mobile device (“*the soul*”), with its sensors and outputs acts as a centralized means to power the prototype and provide the functionality, while the enclosure (“*the body*”) provides the

¹ Portions of this Chapter published in:

Ledo, D., Anderson, F., Schmidt, R., Oehlberg, L., Greenberg, S., & Grossman, T. (2017). Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2583–2593. doi: [10.1145/3025453.3025652](https://doi.org/10.1145/3025453.3025652)

object with meaning both from a visual standpoint, as well as by featuring the means of interaction (e.g. buttons). Specifically, this chapter seeks to answer the first research question posed in Chapter 1:

RQ1. How might designers repurpose mobile devices to prototype smart interactive objects?

The foundation of Soul–Body Prototyping acts as a starting point to then create prototyping tools that enable the authoring of interactive behaviours. I begin this chapter by distilling the designer challenges learned from the first two chapters (§4.1). To address these challenges, I derive a design rationale with key points that drive this dissertation research (§4.2), which then are operationalized through Soul–Body Prototyping (§4.3). I present the design space of Soul–Body Prototyping, showcasing the different dimensions for input and output, power and connectivity, and lastly, how to modify inputs and outputs (§4.4). I show how existing work also follows some of the principles outlined in this chapter.

6.1 MOTIVATION: DESIGNER CHALLENGES

Chapter 1 describes many of the challenges faced by interaction designers today, which can be described as follows:

Need for multiple specializations. Interaction design is a fairly young discipline, which often results in people from other areas (e.g. graphic design) fulfilling an interaction design role. Given requirement to create interactive behaviour (Cooper et al., 2014), the only way to achieve behaviours is through coding. This means that interaction designers are working against their typical training and strength and investing a high amount of time and effort into a medium that (1) they

are not well-versed in, and (2) where the building blocks have not been created to facilitate their quick explorations. In such a case, the consequence is that due to time and budget constraints? designers have to resort to lower resolution sketches, or to work with the tools that they have available, thus creating wireframes in tools such as Adobe XD. This problem is further exacerbated if the prototype needs to be a physical prototype (such as a smart interactive object). To create higher resolution prototypes, the designer would require to create custom circuitry, program it to realize the interactive behaviours, and then place the circuitry into a custom form that looks or feels similar to a final version of the physical product. These are all highly specialized skills which would be unreasonable to expect from a single interaction designer. Collaboration with other experts might be one way to address this challenge, but not one that makes sense if the goal is for the designer to explore varieties of what? and discover nuances in the creation process. In fact, it achieves the opposite, stripping the designer of their agency to discover.

The democratization of graphic design software proves how software tools can disrupt and shape an entire industry. I believe that a similar result could be achieved in interaction design, if the tools can allow designers to come up with functional prototype representations. Graphic design was a discipline in which designers had to create a lot of material by hand (including high quality rendered fonts), which was very time consuming. Different pieces had to be cut out, arranged and then delivered to printing professionals who would screen print it and create duplicates. Graphic design software removed the need to create things by hand and took away the dependency on print spe-

cialists, time delay from waiting and difficulties iterating on the design once it had been printed, as it was already a high time and money investment. Now with graphic design software designers could create and vary their designs and in turn could go from start to finish in creating a high-resolution prototype. Then, the final product became a matter of taking the latest prototype, applying finishing touches and creating all the specifications for printing in the form of a brief. Thus, one way to reduce the need for specialization is to have the right tools in place for people to achieve their tasks.

Lack of Tool Support. Design tools are limited to desktop-based interactions, which favour simple state transitions. While a lot of interactions can be summarized in simple storyboards as trigger-action pairs (e.g. tapping a button to go to the next screen), interactive behaviours are much more nuanced (Myers et al., 2007). People perform actions beyond clicking or tapping (e.g. using fingers to pinch, flick or swipe, or tilting the device altogether), and nowadays can even use physical gestures or even speech input. Consequently, system design needs to distinguish the visual structure/form from the behaviour and interactivity taking place. Systems require more fine-grained real-time feedback for people to understand the effect of their actions, and believe that the system appears responsive and alive, as well as designed with care. A lot of these interactions today only seem to be possible to accomplish through coding, as described in Chapter 4 when I outlined the different kinds of prototyping tools.

Need for Close-to-Product Representations. Given that prototyping is a learning and discovery process in which designers learn by doing, the more they can approximate a close-to-product representation, the

more variety of questions they can probe. More specifically, the continuous responses of an interface or an interactive object can only be assessed if the designer can touch and hold the object/interface. Holding and manipulating the object or interface itself allows designers to subjectively determine if the interaction is successful or whether it needs further nuance. Designers can also give the prototype to others and see how people respond to the authored behaviours. This is of particular importance with behaviours that go beyond trigger-action, given that elements of awareness and feedback rely on animations, sounds or lights that may play as the user performs the action: animations are driven by the interactions. Moreover, the close-to-product representation allows the designer and the rest of the team to see the interaction in context.

To exemplify the extent of these challenges, consider a smart speaker with a digital assistant. When a person calls the smart speaker, the speaker may light up to communicate that it is currently waiting for a command or question (e.g. “what time is it in Caracas?”). Once a command is issued, the speaker may change its pattern to convey that it is now searching online for a response and to acknowledge that it has not simply shut down. The lighting pattern (and perhaps even colour) change again once the speaker is answering, where the brightness of the light may be synchronized to the speech emitted by the speaker. The light colour may reflect the speakers’ confidence or state (e.g. turning red if it is unable to find the answer). Perhaps the speaker features buttons that light up when it is picked up, and specific lighting patterns or sounds may execute once the person presses the different buttons to provide feedback.

In terms of specialization, a designer would not be able to create a physical prototype to try out the potential smart speaker. In fact, the representations would likely be limited to sketches and storyboards that do not reflect the nuanced behaviours taking place simultaneous to the actions. A savvy designer might resort to a video editing tool to try and create a video that carefully reflects all the small transactions as well as the live feedback for a particular scenario. However, there is no way to make a physical prototype that can show the behaviours in context without resorting to creating custom circuits and programming. Thus, the designer has to rely on other specialized team members whom each have their own tasks. This added reliance on other team members brings additional strains to the design cycle: the designer has to wait for the prototypes to be realized, and the designer cannot learn by doing, they can only try out the current version they are provided, only being able to suggest changes and revisions.

6.2 DESIGN RATIONALE

One theme reflected in the literature and often discussed by the HCI community is that *one cannot achieve interaction design complexity without coding*. While it is true that coding can be used in a truly flexible, malleable and expressive manner, prototyping tools can provide higher levels of programming without the need for code and probe specific aspects of the interaction design process. Given that tools afford different building blocks and vocabularies, they each influence the types of solutions that are possible, as well as the most suitable way of thinking to generate such solutions. Thus, I believe that there is a need to explore a broad variety of prototyping tools and integrate

them to work together to: (1) help address some of the challenges present in interaction design practice, as well as (2) reach broader audiences and support different ways of thinking.



In my thesis work, I explore a specific subset of these types of tools in the context described in §4.2, ones that can support *exploration* of – and ability to *try out interactive behaviours*, while still preserving some of the *structural elements* of the prototype. I apply this to smart interactive objects, where the physical form also plays an important role. To do so, I operationalize the aforementioned challenges through the following design decisions:

Enabling Close(r) to Product Representations. To be able to truly try out a behaviour, a representation that is closer to product helps in that designers can tell the nuances of the experience and see how the behaviour plays out. This means being able to work with mock-ups that feel like the real thing – in terms of forms, visual elements and the interactions with them. This also means extending prior work in working beyond mouse-based interactions or wireframes, and extending to other areas of physical computing and mobile interaction.

Minimizing the Need for Coding. Given designers' varied background the expectation of them needing to code is one that needs mitigating. Moreover, if the tool does not require coding, the designer can focus on the design activity itself and not having to think about

more intricate programming constructs that can interrupt the process, as well as consume time and effort.

Eliminating Circuit Building. Circuit building can become even more complex than coding given that the building of the circuit and its programming go hand-in-hand. If the designer wants to create a physical prototype for a smart object, circuit building is a requirement. In fact, Booth et al. (2016) show some of the learning barriers in physical computing, and more importantly how circuit bugs often introduced more software bugs. According to Booth et al. people find it hard to tell if the system does not work because of the circuit or because of the code, often leading people to think it is due to programming issues when in fact it was because of issues in the circuitry. As a result, eliminating the need to collect different components, mount them on a board through soldering and programming them can simplify the process as a whole.

Enabling Authoring of Interaction-Driven Animations. Trigger-action behaviours, such as clicking on a button to go between screens can be easily prototyped in a sketch or storyboard (Myers et al., 2007). Yet, a fundamental component of interactive systems is the continuous response as a result of a person's action. In the digital form, this can be exemplified through interactions such as pinch-to-zoom, where contents of the screen can change size (and progressively show or hide information) as a function of the user's distance between fingers. A video game character may transition from standing, tip-toeing, walking or running depending on the amount of motion applied to a controller's joystick. Lastly, in the physical world these become even more important given the increased amount of

sensing a device has: being able to know if a smart speaker is effectively listening to a command, showing feedback when an oven's knob is physically turned, or even how a smart lightbulb might progressively increase brightness as a person walks into a room. These types of interactions all entail rethinking animations with a new kind of abstraction: animating as a function of interaction (*animation while actions are happening*) as opposed to an abstraction of time (*animation after the action happens*).

Live Changes on Prototypes. Given the focus on being able to try things out, designers should be able to manipulate the prototypes in a plastic manner: change them and fine-tune them as they create them. Ideally, designers can author the prototype and not have to switch an interface or mind-set between editing and testing, which is often a shift that takes place when programming: one way of thinking is required to author a behaviour, then the code needs to be compiled and ran, and then finally the designer can try out the behaviour.

6.3 SOUL–BODY PROTOTYPING PARADIGM: THE MOBILE DEVICE AS A PROTOTYPING ENGINE

To address interaction designers' challenges and achieve the aforementioned goals, this thesis proposes using mobile devices in place of custom electronics, and creating fabricated passive 'enclosures' that define the object's form while exposing the necessary inputs and outputs. Designers can moreover leverage and repurpose mobile sensors and outputs in new interesting ways. A mobile device such as a phone or a watch can act as the '*soul*' to a temporary fabricated '*body*' that holds the form and functionality of a smart interactive object. The

Nintendo Labo² (**Figure 6.1**), which was released in 2018, is an example of a commercial application already applying some of these principles, where the Nintendo Switch's motion controllers and tablet can be enclosed in a cardboard structure to create new interactive experiences. The cardboard shape provides the necessary cues on how to hold the object and provides input mechanisms (e.g. buttons). Mobile devices are well suited as a prototyping engine, as they are ubiquitous and readily available and make for an excellent temporary stand-in for prototyping different smart objects.

While a concern is the high cost of mobile devices – there are two prime reasons as to why Soul-Body Prototyping is feasible. The first is that as part of a prototyping process, the mobile devices are only intended for use temporarily, meaning that only one (or a few) might be necessary to test a concept. Second, given the current technological advances, some mobile devices can be purchased with a low budget (e.g. \$35). Some of these lower cost devices only provide a few sensors, but most seem to provide the basic: touch display, microphone, and accelerometer. Alternatively, a designer can use their own phone or watch, or repurpose an older phone or watch. Moreover, newer prototyping technologies are also adopting these ideas of becoming self-contained devices with basic sensors and outputs. M5Stack³, shown in **Figure 6.2**, uses Arduino technology in a modular case resembling a smart watch. It consists of a base module containing a speaker, touch display, USB port, SD card slot and buttons.



Figure 6.1. Nintendo Labo from 2018 enables players to insert the Nintendo Switch tablet and controllers into cardboard enclosures. Different controller sensors (e.g. the IR camera) are used to sense user input as defined by the enclosure.

² Nintendo LABO <https://labo.nintendo.com/> – last accessed February 2020

³ M5Stack <https://m5stack.com/> – last accessed February 2020



Figure 6.2. M5Stack device. The image shows the M5Stack base module featuring a 2 inch x 2 inch touchscreen, 3 buttons, USB port, and a Lego-compatible peg at the bottom to attach other modules. The bottom three images show the modules for battery, prototyping and GPS. Image adapted from <http://www.m5stack.com/>

Additional modules (e.g. battery, GPS, prototyping board) can be stacked using a Lego-like connection. While many mobile devices share the basic sensors and outputs, it is important to note that more expensive devices will feature newer sensors (e.g. iPhone X's depth sensing front camera) or higher quality parts (e.g. brighter displays with higher resolutions and lower latency).

The benefits to using mobile devices instead of custom electronics in this context are following:

Extensible Geometry. Mobile devices are fairly standardized in their form. Designers can now work with a flat cuboid shape and build more complex forms around it, where the phone or watch becomes the ‘soul’ of the smart interactive object prototype.

Rich Sensing and Outputs. Mobile devices are self-contained, and house a myriad of sensors and outputs present in many smart objects (e.g. touchscreen, microphone, accelerometer). Designers can use or repurpose sensors in interesting ways. For example, designers can use a diffuser or light pipes to create light sources from the phone screen or the camera flash, or use conductive materials (e.g. copper tape) to move the location of a touchpoint to a new location in the physical enclosure (the *body*).

Access to Complex Functionality. Mobile devices also feature internet connectivity, which adds further opportunities for Internet of Things applications. These include voice recognition, as well as web APIs (e.g. weather and Twitter). This also means it is possible for mobile devices to communicate with other mobile devices or smart objects within the larger ecology of devices.

Low Threshold and Less Technical Hurdles. Replacing custom electronic circuits with mobile devices dramatically lowers the threshold for entry. Working with a mobile device removes the need to solder and embed components onto the form, and mitigates the concurrent and tricky ‘circuit vs. code’ debugging previously discussed. Moreover, given the computational power of mobile devices, there is a decreased need for low level programming constructs, such as memory management.

6.4 DESIGN SPACE OF SOUL-BODY PROTOTYPING

Mobile devices are equipped with different kinds of sensors and outputs that provide users with means for direct and implicit interaction. While the output space of mobile devices is fairly simple (i.e. only encompasses touchscreen, speakers and vibration motor, and sometimes could consider the camera flash), the variety of sensors is much higher. In the context of software development, a sensor might refer to the physical electronic component (e.g. accelerometer). Sensors can also refer to a virtual abstraction combining different physical sensor readings (e.g. orientation sensor, resulting from combining information from the accelerometer, magnetometer and gyroscope). Developers can access these different sensors through events⁴.

⁴ In programming, *events* refer to functions that execute once an input takes place.

There are sensors that are treated more independently in software development given their common use and higher relevance, these include: the touchscreen, camera, and microphone. These sensors tend to provide richer information compared to more simple sensors and thus may follow different programming paradigms. For example, touchscreen events might be directly attached to a user interface element (e.g. a button) or might also be accessible via an event. Another example is the camera, where some libraries might provide raw camera images once they arrive, while others call the camera application to retrieve a photo (e.g. Android through its intents platform).

Scope. In this review of mobile sensing and output, summarized in **Table 6.1**, I will focus on common physical sensors (i.e. excluding virtual sensors such as the pedometer, which is simulated via the accelerometer) as outlined in current APIs including iOS⁵, Android⁶ and Windows⁷. I am excluding virtual sensors given that (1) these sensors abstractions in software are inconsistent across operating systems, and (2) one can achieve these virtual sensor readings in custom software, thus they can be brought as a building block where relevant. In addition, the camera, while an important physical sensor, is explicitly excluded from this design space for several reasons. First, cameras have a wide exploration as done in fields of computer vision, and can realize a large amount of sensing through mathematical operations,

Accessed March 2020:

⁵ <https://developer.apple.com/documentation/coremotion>

⁶ https://developer.android.com/guide/topics/sensors/sensors_overview

⁷ <https://docs.microsoft.com/en-us/windows/uwp/devices-sensors/sensors>

creating a very large scope. Moreover, the way in which the camera operates across operating systems is not yet standardized or consistent – in fact, the raw camera image can in some cases be very difficult to access. Lastly, the camera is often excluded from prototyping platforms given that it can be more expensive, complex, and considered “sensitive” given the privacy concerns it can raise. Laput et al. (2017) demonstrate that many of the other physical sensors can be used together and achieve similar versatility and accuracy as the camera. I also exclude the GPS sensor, given that in the context of prototyping, one would need to significantly change their physical location to create anything meaningful with such location data.

The goal of this section is not one of enumeration of sensors but integration of information. I bring in the aspects of both technical development, as well as discuss some applications carried out in HCI. This informs opportunities of what prototyping tools could support, and also helps understand the existing building blocks from the implementation phases that then become accessible to designers so that we, as researchers, can also think of what might become building blocks in the tools of tomorrow. Additionally, the takeaway may vary depending on the reader – while many technical readers may know some of this information, I have found in my experience teaching HCI courses that some elements, such as the fact that accelerometers provide orientation information, are often not known by computer science graduates.

6.4.1 MOBILE SENSORS

Capacitive Touchscreen. The primordial feature in mobile devices for over the last decade is their large capacitive touch screens. Direct

SENSOR	WHAT DOES IT SENSE?	VALUES TO WORK WITH
Accelerometer	Motion Acceleration Basic Orientation (e.g. portrait vs. landscape)	Acceleration X, Y, and Z Gravity X, Y, and Z Magnitude Motion patterns (e.g. shake)
Gyroscope	Rate of change of orientation	Gyroscope X, Y and Z
Magnetometer	Changes in earth magnetic field Compass direction	Magnetic Field X, Y and Z Magnetic Field Magnitude Magnetic Field Direction Compass Angle
Ambient Light Sensor	Brightness	Light value
Microphone	Sound	Amplitude (loudness of a sound) Frequency (pitch of a sound) Sound Pattern Speech (via Speech APIs)
Touchscreen	Touch input on screen	Pointer ID Pointer X and Y Position Contact State (Down, Moved, Up) Gesture (Touch Path Pattern) Contact Area Contact Angle
OUTPUT	TYPE OF STIMULUS	VALUES TO WORK WITH
Touchscreen	Visual	Images Videos Colours Brightness
Speakers	Auditory	Audio Type Audio Volume
Vibration Motor	Tactile	Vibration duration Vibration pattern

Table 6.1. Table summarizing common basic mobile device inputs and outputs

touch is the main way in which people interact with mobile devices, where they can tap on different parts of the screen to transition between different states of the interface. Users might also be able to use continuous actions while touching the display such as pinching or sliding, which can continuously modify the contents of the display. The display might also feature specialized widgets such as buttons or sliders which suggest specific means of interaction. Alternatively, interactions can be more direct and actually operate directly on the virtual object of interest, such as dragging and dropping an object.

Touch interaction also provides different kinds of gestures such as drawing explicit shapes to trigger a command (as shown by e.g. the \$1 gesture recognizer (Wobbrock et al., 2007)), or through variations on the tapping gesture (single tap, double tap, press and hold). Some touchscreens may provide additional information such as the contact area and orientation (Moscovich , 2009; Boring et al., 2012), which can be used to provide additional richness to the input and seamlessly alternate between different modalities, such as previewing versus action (Moscovich, 2009) or panning versus zooming on a map (Boring et al., 2012). Wang and Ren (2009) provide a more in-depth discussion on the input space of the finger on a touchscreen.

Accelerometer. As the name suggests, accelerometer provides readings on the acceleration of a mobile phone in three-dimensional space, thus acting as a motion sensor. Since the accelerometer obtains a reading resulting from the force of gravity and its direction it can provide information on the orientation. For example, one can know if a mobile device is in portrait or landscape orientation and arrange the content accordingly (Hinckley et al., 2000). Orientation can also tell a mobile device when to dim a screen. For example, phones can know when they are placed faced down (e.g. against a table) and can then reduce the level of notifications, while smartwatches will show the time when the wrist is flicked towards the person's head (upright position) and otherwise dim the screen. The accelerometer can provide other means of context sensing through its motion readings. For example, mobile devices can sense a deliberate shake gesture – Apple's

iOS will trigger an *undo* command when shaken⁸. The accelerometer is also often used in synchronous gestures (Hinckley, 2003), where more than can match sensors on more than one device to determine an action, such as knowing when two devices are bumped against each other. If a smartwatch is attached to a person's hand, it is possible to know when that hand interacts with another interconnected device through the accelerometer reading as shown by Chen et al. (2014), Hinckley et al. (2017) and Horak et al. (2018).

Gyroscope. The gyroscope measures orientation and provides a differential (delta) value for three dimensions. The gyroscope is often used in mathematical transformations combined with the accelerometer and magnetometer. The reason for these combinations is that gyroscope information is often integrated over time to obtain rotation information, but often gyroscopes have noise and drifts in their data that introduce errors over time and need to be compensated⁹. Thus, the gyroscope alone is perhaps not directly useful for designers.

Magnetometer. The magnetic field sensor monitors changes in the earth's magnetic field. Thus, the magnetometer can provide some positional information about the device. For example, it is possible to read the magnetic field with respect to the earth's magnetic north and thus determine the cardinal direction of the device, thus acting as a compass. With the three-dimensional reading, it is possible to also get

⁸ <https://developer.apple.com/design/human-interface-guidelines/ios/user-interaction/undo-and-redo/>

⁹ https://developer.android.com/guide/topics/sensors/sensors_motion#java

additional orientation information by integrating with the accelerometer and gyroscope. Additionally, given the magnetometer's ability to read magnetic field, it means a mobile device can sense the presence, strength and direction of a nearby magnet. Some rudimentary applications of the magnetometer with a magnet include Google Cardboard, where a magnet in the cardboard lets the mobile phone know that the enclosure has been closed; and phone cases with covers, where the cover contains a magnet that tells the phone when to turn off the display. In fact, many Android devices will turn off their screen by default when sensing a magnet on a particular location.

Ambient Light Sensor. The ambient light sensor is often placed near the front edges of the display. While it can achieve very sensitive readings and changes in overall lighting, very nuanced readings can be error prone. The ambient light sensor is used as a rough proximity sensor to know when a person holds the phone against their ear (thereby dimming the display), or to adapt the screen brightness to the environment.

Microphone. Mobile devices today typically contain two microphones which are fused into a single signal. Microphone information is collected via sampling, where multiple amplitudes are captured in an array over a very small fraction of time (defined by the microphone's frequency). The array can then be interpreted to obtain either the overall amplitude (or loudness) of the signal, as well as the actual audio captured. One can also operate on the array to obtain the audio frequency (pitch) or use services such as speech recognition to retrieve different kinds of voice commands.

6.4.2 MOBILE OUTPUT

Current mobile devices are also equipped with a variety of output modalities, which designers can leverage in their prototyping. How well the output is provided depends on the *body* which encloses the device.

Display. Most devices are equipped with a high-resolution, full-colour display, and are able to render text, images and video. When creating an enclosure, it will need to expose the display to provide access to this modality.

Speakers. Many mobile devices contain one or more speakers with the ability to output both human-perceivable audio, as well as ultrasonic frequencies. This can be used to prototype objects that provide audio feedback for interactions, notifications, sound effects, as well as emitting speech.

Vibration. Small, vibrating motors are embedded into most commercial mobile devices to allow them to provide haptic feedback. Depending on how the mobile device is embedded into the target *body*, the tactile information can be passed onto the entire target object, or localized areas. This feedback can be used to enable discreet notifications, or direct feedback to user operations.

6.4.3 POWER AND CONNECTIVITY

Connectivity. Currently, most mobile devices have a variety of wireless capabilities, allowing them to connect to the internet and other devices via Wi-Fi and Bluetooth. With this connectivity, devices are able to retrieve information about current events, the environment, and a wide range of other information that the user or system can act on. Thus, any prototype is wireless and web-enabled.

Power. A core requirement of the vast majority of interactive devices is power. Mobile devices are by default equipped with a battery, as well as a connection port to recharge them. Moreover, one can also use an external power source if the prototype needs to be used for more extended periods of time.

6.4.4 INPUT AND OUTPUT MODIFICATION

In addition to using the sensors of a mobile device for their intended purpose, my work extracts an additional layer of inputs and outputs through examining the existing literature. I refer to these as *modifiers*. Modifiers expand the sensing and output capabilities of the mobile device. Modifications can take the form of *rerouting* modifiers, or *transducing* modifiers.

Rerouting Modifiers

Rerouting modifiers preserve the nature of the sensing or output, but change the location where it takes place. Rerouting can be particularly useful if input and output is desired on different locations of the target object, which extend beyond the physical size of the mobile device. For instance, light pipes can be used to reroute light output to different physical locations as done by Savage et al. in PipeDream (2014).

Input can be rerouted in a number of ways, for instance, capacitive touch sensing can be moved to off-screen buttons using conductive pathways as done in Clip-On Gadgets (Yu et al., 2011), Extension-Sticker (Kato et al., 2015), and Midas (Savage et al., 2013). Touch sensor can thus be placed in a more convenient or appropriate location. Similarly, light pipes can be placed over the ambient light sensor to sense lighting in remote locations of the physical prototype.

Transducing Modifiers

Transducing modifiers are those which allow for different types of interaction that extend beyond the base capabilities of the mobile device. This fosters a new set of interaction modalities on. For instance, Acousturments (Laput et al., 2015) leverage the microphone and speakers to sense how tangible controls are being manipulated, thus adding novel input mechanisms to a mobile device. Microphones can also enable gestural or tangible interactions around the object, as done with Lamello's custom 3d printed widgets (Savage et al., 2015), or by detecting the discrete audio patterns when interacting with the object or the environment and responding accordingly (Lopes et al., 2011; Harrison et al., 2011).

Beyond audio, the magnetometer can be used to sense the movement of a magnetic field as input from buttons, such as with Nenya (Ashbrook et al., 2011) and Google Cardboard¹⁰. One can create different kinds of physical widgets such as switches and sliders with embedded magnets as done in MagGetz (Hwang et al., 2013). Alternatively, one might sense the intensity or the orientation of the magnet (Hwang et al., 2013b; Cheung et al., 2019).

The vibration motor can be used to move the prototype, as demonstrated by the Nintendo Labo, where a cardboard bug toy moves through the vibration of two attached game controllers.

These are only some examples of ways in which sensors and outputs can be repurposed. By integrating the types of sensing and knowing the ways in which they can be repurposed, designers are conceptually

¹⁰ <https://arvr.google.com/cardboard/> Accessed February 2020

equipped with new ways of designing smart object prototypes that leverage mobile devices and their inputs and outputs.

Modifiers are overall powerful prototyping tools, as they can push the inputs and outputs beyond the touchscreen of the mobile device. Increasingly, both industry and research continue to explore potential avenues to provide new inputs and outputs with the currently available sensors. Sensors such as the touchscreen, accelerometer and microphone are unlikely to go away. Over time, new capabilities may become available to mobile devices. For example, newer phones such as the Google Pixel 4, have already included radar sensing technology, which is low cost and shows promise in sensing not only physical gestures, but also an ability to detect and classify a variety of objects, as demonstrated by RadarCat (Yeo et al., 2016). In that sense, the design space of Soul-Body Prototyping is likely to grow as these sensors become commonplace.

6.5 CONCLUSION

This chapter presented the Soul-Body Prototyping paradigm, which is a metaphor that proposes using mobile phones as a '*soul*' which gets inserted into a physical '*body*' which acts as a temporary enclosure to simulate a physical prototype with meaningful form and exposing only the necessary inputs and outputs. The ideas behind Soul-Body Prototyping are compatible with any sensing technique that can be used with a mobile device sensor which fosters a variety of novel inputs and outputs that can be used. Moreover, given that many commercial products and research projects already have applied these means of interaction, it shows that is feasible and has the potential to become common place. The next question is whether Soul-Body

Prototyping as a paradigm is something that can be adopted easily, and how to make it to bring it as close as possible to a walk-up and use paradigm for designers as well.

CHAPTER 7. SOUL-BODY PROTOTYPING CASE STUDIES

Soul–Body Prototyping is one way of *repurposing existing hardware and software to enable designers to prototype interactive behaviours for smart interactive objects*. It is particularly feasible, as mobile devices are common place and can act as temporary placeholders for prototypes. As I progressed through my dissertation research, the ideas behind Soul–Body Prototyping evolved, and additional products and projects emerged. From the early stages, however, it was possible to see a lot of the expressive power the mobile device provided through its portability and also through the sophisticated inputs and outputs. This chapter¹ presents case studies that demonstrate the feasibility,

¹ Portions of this chapter have been published in:

Hung, M., Ledo, D., & Oehlberg, L. (2019). WatchPen: Using Cross-Device Interaction Concepts to Augment Pen-Based Interaction. *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, 1–8. doi: [10.1145/3338286.3340122](https://doi.org/10.1145/3338286.3340122)

flexibility and expressive leverage provided by Soul-Body Prototyping. I first describe early student explorations in an advanced HCI undergrad course conducted in 2015 (§7.1), where students in a short time-span used electronic components and mobile devices to prototype novel smart objects. Given that there were external electronic components in these early student prototypes, I discuss how these solutions could be migrated into Soul-Body implementations that do not use any external electronic components beyond the mobile device. I then move on to discuss WatchPen (§7.2), an undergraduate research project I supervised, in which the student successfully applied Soul-Body Prototyping to create a smart stylus that connects to a tablet device, and can handle a variety of sensors at once.

Undergraduate computer science students learning design as participants act as a first-step exploration of what is possible with the Soul-Body Prototyping paradigm. They represent an initial hypothetical case of what might happen if designers did not face the core three challenges presented in the last chapter. While unfamiliar with the topic, students were able to accomplish expressive and rich prototypes in short timespans of five weeks and three months. As a result, it shows promise that it should be possible to use Soul-Body Prototyping to (1) continue to push HCI research forward; and (2) enable interaction designers to use Soul-Body Prototyping, with the right tools, to create a wide variety of prototypes. More details and limitations will be discussed in Section §7.3.

A video figure for WatchPen can be found at: <http://davidledo.com/projects/project.html?watchpen>

7.1 EARLY STUDENT EXPLORATIONS

During my early explorations of Soul-Body Prototyping, I asked students in a second HCI computer science class (CPSC 581) to create smart object prototypes using mobile devices and electronic components. Students worked with Windows Phones (Nokia Lumia 735), programmed in C#, together with Phidgets components (which included a kit with sliders, pressure sensors, servo motors, buttons and others) connected to a desktop computer (also developed in C#). The electronics were to be placed on top of- or around the mobile device in a new custom form. To connect the electronics and mobile devices, I created a networking library² and set up a relay server that would enable students to easily send messages between the mobile application and the desktop. Students had five weeks to create a prototype, and had to deliver prototypes during different stages: 20 concept sketches and variations, a non-functional form prototype using materials such as foam core and paper, a first demo to test the implementation and a final prototype demo and video. Students posted the results in their web portfolio and were asked for permission to discuss their projects with attribution once the course was over.

From 15 students, I have selected 5 representative prototypes that exemplify interesting Soul-Body Prototyping applications. These show (1) an interesting breadth of early ideas that were developed and functional; (2) the feasibility for computer science undergraduates taking their second HCI course to apply an early version of Soul-

² Kevin Ta formalized the library into a Github repository for a future installment of the course: <https://github.com/kevinta893/NetworkIt>

Body Prototyping. Indeed, there are a few factors to keep in mind. First, these students are trained programmers with little design experience, who are working with electronic components (albeit not requiring to solder or perform major circuit building). Second, the class setting is an artificial setting, where students are working towards earning a grade. Lastly, the use of electronic components means that there was little use of the built-in sensing and output of the mobile device, and that potentially some components (e.g. servo motors) could not have an equivalent Soul-Body Prototyping counterpart.

I describe the selected prototypes, and show how they can be converted into the current paradigm of Soul-Body Prototyping.

7.1.1 GYMBUDDY BY MIKE CHUBEY

The Prototype. Gymbuddy (**Figure 7.1**) is a workout assistant which could be attached to any workout equipment via a hook-and-loop (Velcro) strap. The form of the device was heavily padded in case the device would fall or get hit during intense workout activities. Within the padded body, a distance sensor would measure when a gym-goer performed a particular repetition for a workout, such as performing a



Figure 7.1. Gymbuddy (by Mike Chubey) is a mobile device which can be attached to gym equipment (e.g. bench press) and provides training and assistance through the display. The system keeps track of repetitions via a distance sensor.

pull-up, a push up, or using a workout bench. The GymBuddy assistant would guide users through workout routines, keeping count of repetitions and sets, and suggesting new exercises.

Converting it into a Soul-Body Implementation. Because the GymBuddy system primarily uses a distance sensor to sense when a person performs an exercise repetition, there are a few alternative mobile sensors one might use to create a testable version of the prototype. One can use the ambient light sensor to know when the gym-goer is close to the device by looking at the fluctuation of illumination on the front of the mobile device. However, this might prove error-prone, as any change in lighting might trick the system into thinking that a repetition has been achieved. Alternatively, one can use the device's accelerometer to sense the directional motion. The accelerometer solution would work in the case of some workout equipment, such as doing weights on the bench, where the mobile device is attached to the bench itself. In the case of the push-ups, where the mobile device is placed on the floor, there is no way to sense the person's motion with the accelerometer. In that case, perhaps one might have to modify the design itself, and require the user to strap the mobile device to their body (e.g. to their chest or arm) to sense the upwards and downwards motion.

7.1.2 PATHOLOGIST DEVICE BY TERRANCE MOK

The Prototype. One of the students developed a special device for his wife, who is a pathologist (**Figure 7.2**). The device featured a caliper

for measurements: turning a knob would cause a servo motor to operate the rack and pinion caliper. A pathologist could navigate through a human body silhouette on the phone and choose an area to mark, where they could use the device's calipers to measure the mark (e.g. a tattoo), enter additional information, comments, or photos and store them.

Converting it into a Soul-Body Implementation. The only challenge in this prototype is that the caliper in this implementation relies on a servo motor to power the rack and pinion mechanism. One way to address this is to simply move the rack and pinion with an analog knob. To keep a record of the measurements, one can add a capacitive marker that reaches the touchscreen and use the coordinates of that contact point to map the value to the correct measurements.

7.1.3 SMART DOCKS BY ORKHAN SULEYMANOV

The Prototype. Another student developed a set of docking stations which would provide different kinds of physical controls and affordances to the mobile device (Figure 7.3). Placing the mobile device in these stations temporarily transformed the functionality of the phone. One docking station acted as a music station, which opened the music app and provided physical controls to play/pause, navigate through songs or mute. Another station acted as an alarm clock for the bedside table, which could be set, snoozed or dismissed. The phone was augmented with an RFID tag, while the docking stations contained RFID readers to detect when the phone was stationed.

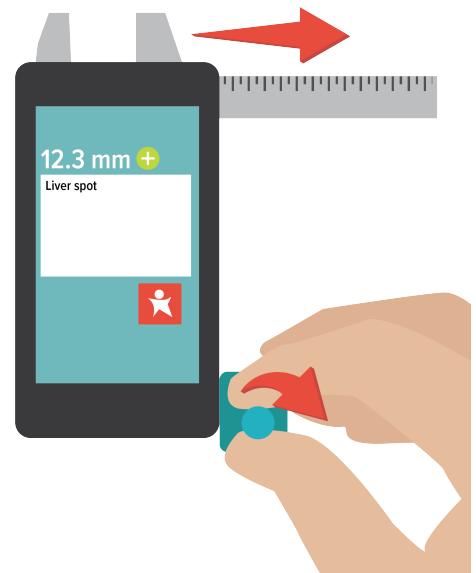


Figure 7.2. Pathologist device (by Terrance Mok). The device features a caliper powered by a knob and a servo motor which is used to record measurements on different body features (e.g. a tattoo or a scar). The application allows pathologists to select different parts of the body to enter the recorded information.

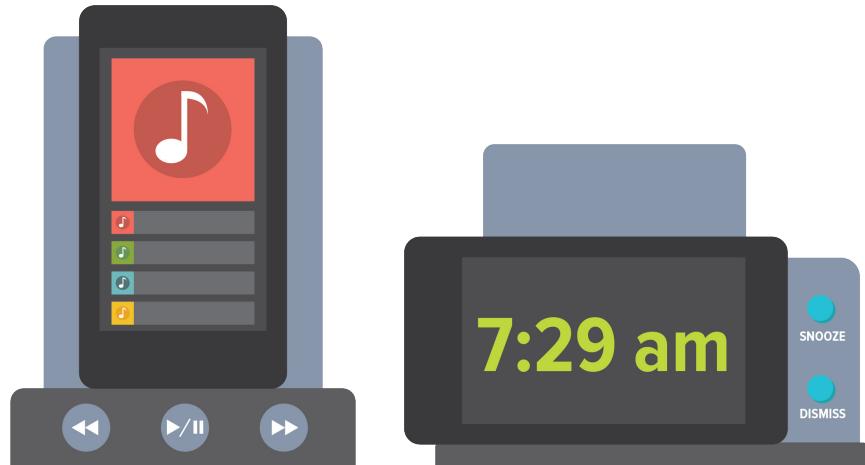


Figure 7.3. Smart Docks by Orkhan Suleymanov provides physical controls at different locations of a home with specialized usage contexts: a music station, as well as an alarm clock.

Converting it into a Soul-Body Implementation. There are two main things that need to be converted in this prototype, one is the need for physical controls, and the second is the ability to recognize the current docking station. For prototyping purposes, one can allocate the magnetometer for one of the two functionalities. For example, the different buttons can have a magnet associated, then the mobile device can read changes in the magnetic field vector orientation changes once one of the buttons is pressed. Alternatively, the docking stations can have different magnet strengths which are then used to recognize the identity of the stations. In that case, the buttons perhaps can have different sound signatures which are then picked up by the device microphone. One might also consider using NFC tags if the phone supports reading these tags, though this functionality is typically reserved for higher end mobile phones.

7.1.4 HUGGABLE PHONE BY SARA WILLIAMSON

Prototype. A stuffed animal holds a phone which performs video calls to parents at a distance? (Figure 7.4). Children can squeeze the stuffed animal's hands to call a parent who might be away due to travel. If the parent was not available, a video greeting of the parent would show up, and then allow the child to leave a video message.

Converting it into a Soul-Body Implementation. The main physical means of interaction at play in this prototype is through squeezing the plushie's hands, which hid a pressure sensor. A quick alternative to this approach is to use copper tape to reroute touch input from the toy's hands to a location on the mobile device touchscreen. If the designer wishes to make a more finished version of this, they can try different forms of conductive fabric and thread that can reroute the information, all while ensuring that the contact points are large enough to be recognized as touches by the mobile phone.

7.1.5 PHOAME SWORDS BY KEVIN TA

Prototype. Phoame Swords is a physical sword fighting game (Figure 7.5) in which players wear a shirt with “*hit points*”. Hitting an opposing player with the sword on a hit point decreases the attacked player's health. The game is over when players' health hit zero, and one remains. The players wield a sword which holds a mobile phone showing the current health points and playing sound effects when a player is hit, or when the sword is swung. The hit points on the special shirt were implemented by having RFID readers in each one. The sword had an RFID tag attached to its tip, so the system could recognize when a player is attacked.



Figure 7.4. Huggable Phone by Sara Williamson presents a stuffed animal hugging a phone. The mobile device can be used to video call a parent who is currently away.

Converting it into a Soul-Body Implementation. This is perhaps the hardest prototype to realize in a full Soul-Body implementation devoid of electronics. The reason behind this challenge is that the distance between the mobile device and the tip of the sword makes it difficult for the mobile device to sense contacts and also preserve the identity of which hit point and which person was hit. Perhaps one could reroute touch points of the screen to the sword's tip (via copper tape or conductive ink), and make the targets also with conductive material, thus instigating touch events on a specific region of the mobile device, which is then used to compute the new health values. This of course assumes that there are only two players, where a more sophisticated implementation would be required to fully experience more complex gameplay.



Figure 7.5. Phoame Swords by Kevin Ta is an augmented physical game in which players engage in a sword fight until they run out of health. The mobile device plays sound effects and keeps track of player's health / hit points.

7.2 WATCHPEN: LATER STUDENT EXPLORATION

WatchPen (Hung et al., 2019), shown in [Figure 7.6](#) was an undergraduate student project led by Michael Hung under my mentorship, implemented over the course of three months as part of an HCI research course in the fall semester. This particular project examined the role of an augmented stylus for interacting with tablet devices and its benefits with added sensing and outputs. More importantly, however its implementation was a Soul–Body Prototype which enabled a comprehensive exploration of what interactions with tablets might look like in the context of a drawing application. Specifically, the augmented pen leverages different smartwatch sensors and outputs, as well as their combinations, to envision possible features that could be added to a tablet stylus, and serves as an example of how Soul–Body Prototyping can be used to carry out in-depth explorations without the need for specialized hardware.

Tool Selection via Accelerometer Posture Sensing. The three-dimensional orientation as sensed by the accelerometer can be used to detect different postures in which the pen is being held. As a result, one can envision how different grips can be used to switch between tools in a drawing application. Given how the user holds the pen, exemplified in [Figures Figure 7.7-b](#) and [Figure 7.9](#), one can switch between: regular brush, airbrush, stamp and eraser.

Display Controls and Awareness. One problem with direct input is that the information about the current state of the tool is dissociated from the location in which the action takes place. With a mouse cursor, it is possible to provide feedforward informing the user of the

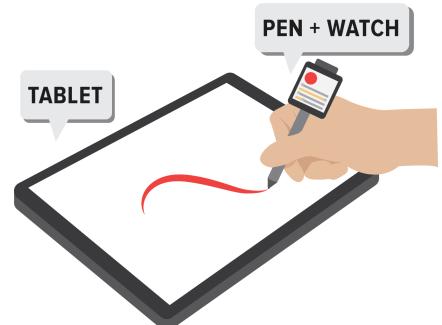


Figure 7.6. WatchPen is a tablet stylus that has been augmented with a smart-watch. WatchPen explores how different sensors and outputs can augment tablet interactions in the context of a drawing application.



Figure 7.8 Physical airbrush (a), and its replication in WatchPen (b) which can control the ink flow with the watch's touchpad. The orientation of the pen (c) changes how the paint is spread on the canvas.

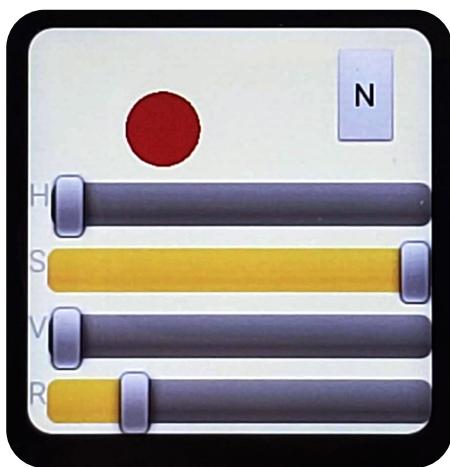


Figure 7.7. The WatchPen display shows the current colour, as well as hue, saturation, and brightness and radius sliders which can be adjusted anytime.

current state reflecting the effect of a click: the cursor can change its appearance and become a pencil and draw or a hand to move the canvas around. The cursor can provide even additional information about the tool's parameters, such as the colour or stroke size. With touch or pen interactions, these details are often relegated to the side of the interface, or hidden within menus. Having a display attached to the pen can help provide additional information about the current tool and its parameters (e.g. stroke size and colour), as well as the ability to change those parameters through different sliders (see **Figure 7.7**).

Orientation-Based Airbrush. With orientation and touch sensing, it is possible to simulate the way a physical airbrush operates (comparison shown in **Figure 7.8-a** and **b**). For example, the finger can touch the watch's screen to control the paint flow (i.e. radius) and the orientation can change how the paint is spread (**Figure 7.8-c**).

Microphone-Enabled Tonal Brush. With the added sensing from the smartwatch, it is possible to explore new kinds of unusual and crea-

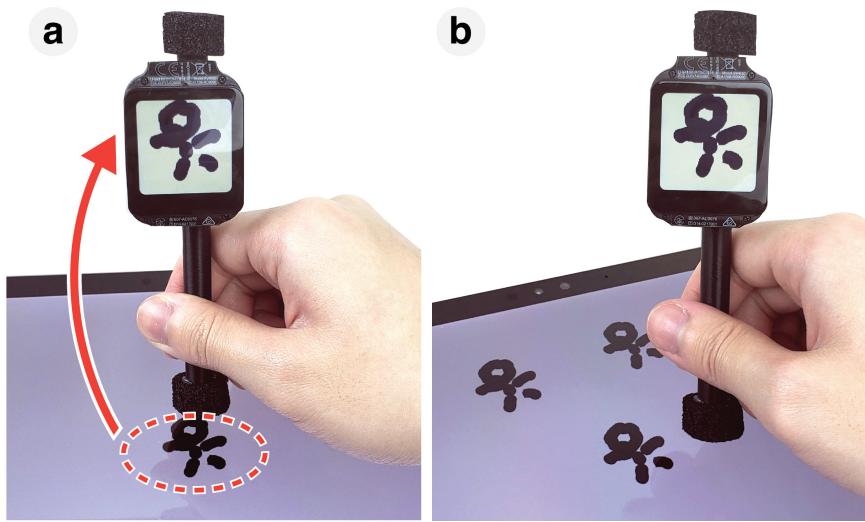


Figure 7.9. Stamp tool in WatchPen, triggered when the pen is held upright. The pen tool can capture contents (a) and show it on the display, and then paste copies on the canvas (b).

tive ways of operating a stylus. For example, it is possible to map different parameters of a brush from the sound captured by the microphone. The sound frequency/pitch can be mapped to different hues, while the amplitude/loudness can be associated to the brush size.

Stamp Tool. Holding the pen perpendicular to the tablet switches to the stamp tool (see **Figure 7.9**), which can be used to apply copy and paste operations. Moving the pen around the tablet captures and area and displays it on the watch screen, which acts as a visual clipboard. The user can then paste the content multiple times by pressing the upright stylus against the tablet.

Tactile Feedback through Vibration. WatchPen also leverages tactile feedback through the watch's vibration motor. The vibration is used to notify the user when a tool has been switched due to a change in posture, as well as when erasing.

7.3 DISCUSSION

Studying how students worked with Soul-Body Prototyping provided me with a first-hand experience on how people might operate the metaphor and paradigm and shows the expressiveness as well as the feasibility. However, the limitations of these explorations, as well as the lessons learned for both HCI and Design contexts provide interesting insight that informs the design of prototyping tools that support Soul-Body Prototyping.

7.3.1 LIMITATIONS

What can we extrapolate from student explorations?

Student explorations are beneficial in that they are one way to explore the feasibility of a concept, especially if it is possible to provide them with the right equipment and tools. It can be argued that students are to some degree novices in HCI and design, since this advanced HCI class shifted focus from methods of inquiry to learn about end-users, to sketching and prototyping using a variety of technologies. This means that the students are essentially beginners in design but have a strong technical ability compared to design practitioners. At the same time, there are a few artificial components to a student-based exploration. The first is that participants are being driven by an assignment that has a grade component to it. Second is that students have access to help at all times from the course instructors, and the assignment is scoped in such a way that it can be solved if the work is put in. In both explorations, I provided students with software libraries that would facilitate the programming experience, so they could focus on the design component. Regardless of these limitations, there are key assumptions that can be made:

- Students were able to apply Soul-Body Prototyping to invent novel smart objects
- Students were able to explore a variety of mobile inputs and outputs and create rich experiences in a short timespan
- Students created forms around the prototypes to give them meaning using a variety of readily available materials such as foam or cardboard

Understanding Sensors

Perhaps the biggest challenge and limitation of these explorations, and a lesson for Soul-Body Prototyping is that people who apply the paradigm must either understand sensors or have some degree of assistance. For example, the accelerometer is used to sense the basic orientation information of a mobile device, yet students did not know the accelerometer data could be used to assess orientation. Similarly, with the microphone data, the mobile device simply returned an array of bytes every sample which needed to be somehow interpreted meaningfully. Working with these kinds of sensors are very different than working with, say, the touch sensor, or a mouse cursor, which have a specialized event where the resulting data can be understood right away through a simple print statement on the console.

Many of the students in the class were programming on the mobile device for the first time. While the programming platform was set up to remain familiar to the students (e.g. using C# and Windows Universal Platform for Windows Phones), the gap was more in understanding what the mobile sensors were capable of and how to make sense of the increasing amount of data. Given the need to make sense

of the array of sensor data in some way, it is perhaps why many prototyping tools have opted to simply not show the available sensor data or make it usable, as sensors beyond the touchscreen could perhaps be simply categorized as niche applications, and get dismissed under the assumption that only experts should delve into.

7.3.2 SOUL–BODY PROTOTYPING AND HCI RESEARCH

The explorations in WatchPen show how it was possible to explore a design space in a short amount of time. While I had provided Michael Hung, the lead author, with a software library that handled the networking and a lot of the sensing, along with examples, he recognized the advantages of working with higher-level programming specific components rather than low-level hardware. I have already discussed in Chapter 5 the power of toolkits as software infrastructures, and using mobile devices instead of hardware components are no different. One example of repurposing devices similar to Soul–Body Prototyping is Lee’s prototypes which he referred to as “*procrastineering*” (2008). Lee (*ibid*) was able to create a variety of prototypes in a short amount of time, such as an interactive whiteboard, a head-tracking VR display and a multitouch display all using a Nintendo Wii Remote and its infrared camera through a widely available software library. In that sense, the infrastructures build on top of one another until they become high-level enough that they can increase people’s ability to participate in the prototyping activity, or people can save time that would have been spent working with lower level components otherwise. This leads to two obvious directions for Soul–Body Prototyping in HCI research. First is to continue exploiting mobile sensors in new

and interesting ways using high-level programming platforms. The other direction is to create additional layers of infrastructure that can simplify the design and development process and open up new interesting *paths of least resistance* (Myers et al., 2000).

7.3.3 SOUL BODY PROTOTYPING AND DESIGNERS

While the participants of these activities are not interaction designers, they are an example of students beginning their training in design who happen to have the technical skills to code interactive applications. The results of these explorations then suggest what could be possible if designers did not have these technical difficulties. Moreover, the prototypes also show that simple materials such as cardboard and foam core are enough to build a basic form and focus on the interactive behaviour implementation.

With the Soul-Body Prototyping paradigm, it is possible to achieve more closer-to-product representations, provided designers can create a physical form of some fidelity and resolution. It removes the need for technical expertise in circuitry and brings a few more tools to their disposal, such as mobile apps or websites which can be recontextualized. However, it becomes clear that there is a need for a new class of prototyping tools that can help design interactive behaviours beyond the WIMP (Windows, Icons, Menus and Pointers) paradigm and shift towards different sensor-based input approaches. While there is still a need to learn and better understand some of the sensors, the tools can help conceptualize some of these challenges.

7.4 CONCLUSION

This chapter presented case studies of Soul-Body Prototyping as a paradigm to circumvent the need for custom electronics to create smart interactive object prototypes. I showed some early examples of computer science students who programmed and used high-level electronics to create rich innovative prototypes over the course of five weeks. This shows that using mobile devices in this new prototyping context can inspire new interesting ideas and lead to fairly sophisticated implementations which demonstrate the paradigm. I then discuss how these implementations can be converted into prototypes that use only built-in sensing on the mobile device. WatchPen (Hung et al., 2019) then shows the power of Soul-Body Prototyping at a larger scale and even demonstrates how a variety of sensors can be brought together. Indeed, these implementations raise the question of how to take this knowledge and operationalize it for designers to create new prototypes, which is the goal of the systems in the upcoming chapters. Chapter 8 shows how a system might use basic behaviours to automatically generate 3D printable forms (Pineal) for Soul-Body prototypes, while Chapter 9 shows a means to author nuanced interactive behaviours with (Astral).

PART 3

SYSTEMS

CHAPTER 8. PINEAL: BEHAVIOUR-DRIVEN PHYSICAL PROTOTYPING

“Form follows function – that has been misunderstood. Form and function should be one, joined in a spiritual union” – Frank Lloyd Wright

The last part proposed the Soul–Body Prototyping paradigm as a method for *repurposing existing hardware and software to enable designers to create live interactive prototypes for smart interactive objects*. In it, designers place a mobile device (*soul*) into a temporary physical form (*body*) to create a physical prototype for a smart object. The paradigm on its own helps alleviate some of the designer challenges, such as the need for specialization and the need for close-to-product representations. However, Soul–Body Prototyping needs tools that enable its operationalization. Specifically, it opens up the second research question stipulated in Chapter 1:

RQ2. How might designers author forms around mobile devices to make them look and feel like smart objects?

To address this research question, I created Pineal¹, a prototyping tool capable of generating a functional smart object prototype. The resulting prototype consists of a fabricated form (e.g. 3D printed) which houses a mobile device which performs all the programmed behaviours. With the aim of providing a base understanding of Pineal and its research contributions, I begin this chapter by providing a summary description of the system (§8.1), while also situating it within the existing related work (§8.2). This then enables describing the more technical details behind Pineal, such as the interface components (§8.3), how a user might leverage these components to create an example prototype (§8.4), and the implementation details that realize the end-user operations (§8.5). Further validation of Pineal and its expressiveness is achieved through a series of prototypes I created using the system (§8.6), which cover the elements of the design space of Soul–Body Prototyping. Lastly, I discuss Pineal’s concept, limitations and reflections (§8.7) which are then tied together with concluding thoughts (§8.8).

8.1 PINEAL

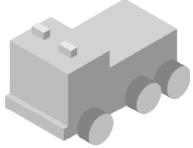
Pineal is a software prototyping tool that operationalizes Soul–Body Prototyping and enables designers to author smart interactive object prototypes. Pineal enables: (1) programming interactive behaviours

¹ Portions of this chapter have been published in:

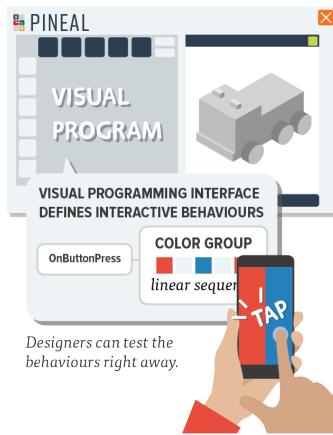
Ledo, D., Anderson, F., Schmidt, R., Oehlberg, L., Greenberg, S., & Grossman, T. (2017). Pineal: Bringing Passive Objects to Life with Embedded Mobile Devices. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2583–2593. doi: [10.1145/3025453.3025652](https://doi.org/10.1145/3025453.3025652)

Video figure: <http://davidledo.com/projects/project.html?pineal>

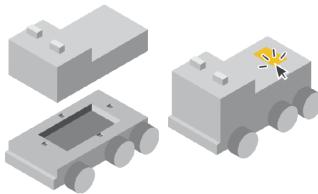
1 IMPORT AN EXISTING 3D MODEL
Designers can import custom or downloaded 3D models in standard formats.



2 PROGRAM INTERACTIVE BEHAVIOURS
Pineal provides a visual programming interface with different smart object inputs and outputs. Behaviours are automatically loaded on a companion mobile device.



3 MODIFY 3D MODEL
Pineal takes into account the programmed behaviours and with the help of the user modifies the 3D model to fit the mobile device and create the appropriate inputs and outputs on the physical form.



4 FABRICATE AND ASSEMBLE
The modified model is exported and ready for 3D printing. Designers can then assemble the parts, insert the mobile device and interact with their prototype.

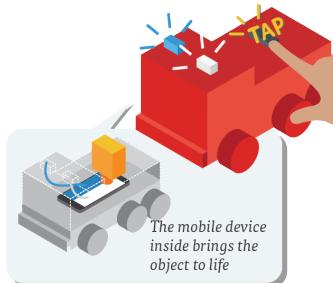


Figure 8.1. Overview of Pineal.

on mobile devices (the *soul*); and (2) modifying a 3D model form to house the mobile device and expose the phone or watch's inputs and outputs (the *body*). As a result, one can use an existing fabrication method, such as 3D printing, to realize the object's form, and insert the programmed mobile device into it to create a new prototype that resembles a smart object.

Pineal features two main workspaces. On the left, there is a visual programming environment which leverages trigger-action modules as node-link diagrams. The right side shows a 3D modeling environment, which operates Autodesk MeshMixer in the background. The 3D modeling environment serves to show the 3D model, and also to perform all necessary automation tasks that will modify the 3D model to make it into a physical form that supports Soul–Body Prototyping. This means that the form can fit the mobile device and facilitate inputs and outputs which connect to said mobile device. Pineal works through four steps, summarized in **Figure 8.1**:

1. Importing an Existing 3D Model. Designers can import any standard 3D model file format (e.g. STL or OBJ) into the 3D modeling environment. As a result, one can work with existing models downloaded from the web (e.g. through a model-sharing site such as Thingiverse.com), or with a custom model created from scratch on any 3D modeling tool (e.g. Meshmixer, Autodesk Fusion 360, SolidWorks, OpenSCAD).

2. Programming Interactive Behaviours for Mobile Devices. The visual programming environment provides a set of building-blocks for trigger-action behaviours that leverage mobile device sensors and outputs. Pineal features a mobile device client which connects to the

desktop application to author interactive behaviours. The design of the visual programming is such that designers can immediately test and modify the behaviours in a live manner: a change in the visual program is reflected on the mobile device in run-time (**Figure 8.1-2**).

3. Modifying the 3D Model. The programmed behaviours send modification instructions to the 3D modeling environment, which the designer can step through to customize the form (**Figure 8.1-3**). For example, if the visual program requires the mobile device screen to be visible (e.g. a module output displays text), the 3D modeling environment will add a step to create a cavity to expose the screen. The designer can then select the location of the 3D model where they wish to create the screen cavity. Once all steps are finalized, the system takes these constraints and automatically modifies the original 3D model: it splits it into two pieces that can be assembled, creates room for the mobile device, and exposes the inputs and outputs specified by the visual program, such as exposing the device's screen in the previously selected location.

4. Fabricating and Assembling the Model. The modified 3D model is exported as a file to any folder, and is now ready for fabrication via a designer-chosen slicer software. The designer can realize the physical forms (e.g. through 3D printing) and place the mobile device. Once the device is inside, the programmed behaviours run as expected, thus temporarily turning the mobile device into a smart object prototype. Interaction designers can use the prototype to envision the behaviour in the context of a physical form that can closely resemble real world use (**Figure 8.1-4**).

8.2 RELATED WORK AND CONTRIBUTIONS

Pineal integrates different technologies to author smart object behaviours. In particular, there are two main benefits over prior work, discussed below.

1. Leveraging a Single Mobile Device Rather than Many Electronic Components. By repurposing mobile devices, designers can have all the primary functionality of a smart object through the myriad of mobile sensors, all contained in an easily extensible geometry. Existing work to date focuses on the creation of custom circuitry. Different platforms exist that lower the thresholds for programming electronics, where the focus is on removing the need for circuit building, but still require end-users to code and create the surrounding physical form. For example, Phidgets (Greenberg & Fitchett, 2001), Smart-Its (Gellersen et al., 2004), and .NET Gadgeteer (Villar et al., 2013) offer a set of premade electronic boards with sensors and resistors that programmers can easily “*plug and play*”, all while coding in familiar object-oriented, event-driven environments, such as C#. As a result, an expert programmer can author physical user interfaces that can sense human input or contextual information from the environment. Other platforms, such as Trigger-Action Circuits (Anderson et al., 2018) and PaperPulse (Ramakers et al. 2015) leverage more inexpensive electronics (e.g. Arduino), provide simplified visual programming, and automatically generate a full circuit diagram to help with assembling the circuit based on the pre-programmed behaviours. In the case of PaperPulse, the circuit diagram can be printed

with conductive ink using a modified printer. Micro:Bit² packs a set of basic sensors (accelerometer, temperature sensor, compass, Bluetooth, and two buttons) and a 5x5 LED display, thus containing hardware building blocks on a single board. Micro:Bit can be programmed with TouchDevelop (Ball et al., 2016), web-based visual programming language that helps novices such as makers and children to author basic behaviours. However, the Micro:Bit was designed to teach children programming, and thus has a few arrays of low-cost sensors and outputs. Thus, while Micro:Bit removes the coding complexity and the circuit building, the range of what can be created with the few sensors and outputs is limited. By tapping onto the mobile device's sensors and outputs, Pineal brings forth a wider variety of programmable sensors and outputs, while accessing more complex functions such as web operations through its internet connection, or built-in services such as speech recognition.

2. Using Programmed Behaviours to Automatically Modify 3D Models. Creating 3D models from scratch is difficult, especially when those models have to match existing, real-world geometries. A number of projects have examined integrating electronic components with 3D models. For instance, RetroFab (Ramakers et al., 2015) automatically generates enclosures for electronic components which are used to actuate existing physical interfaces. PipeDream (Savage et al., 2014) allows users to easily author internal pipe structures within 3D printed objects, which can be applied to a range of inputs

² <https://microbit.org/> – Accessed December 2019

(capacitive sensing) and outputs (haptic feedback, light pipes) techniques but relies on the author to define the location and purpose of these pipes. MakersMarks (Savage et al., 2015) allows users to physically sculpt and create an object by hand using clay and use tags to specify physical input mechanisms. Objects are then 3D scanned, and the model combines the tagged markers to create a solid model that reflects both the overall form as providing cavities to fit the electronic components. Enclosed (Weichel et al., 2013) is a modeling environment to create enclosures for electronic objects. While it allows for custom shape creation, its focus is on incorporating the shape of the electronic components, not on working with existing 3D models. MixFab (Weichel et al., 2014), CopyCAD (Follmer et al., 2010), and KidCAD (Folmer et al., 2012) all allow novices to begin to perform 3D modelling operations using real-world objects. The objects are scanned in via cameras or sensors and brought into a 3D modelling environment which simplifies the modelling process. In many of these approaches, the modelling techniques are simple enough for novice end-users to use. Pineal builds on these prior works by automating the 3D modelling tasks necessary for embedding devices into 3D objects. Pineal leverages prior algorithms such as creating internal pipe structures to route fiber optic cables (Savage et al., 2014), while introducing new functionality such as automatic splitting of 3D models to physically insert mobile devices inside the form. Moreover, this automation is entirely dictated by the behaviours specified by the designer, thus reducing the need for manual 3D modeling operations while still providing customization opportunities. As a result, users technically do not need much understanding of 3D modelling, thus simplifying the creation of smart object forms.

8.3 INTERFACE AND WORKSPACES

The system contains a visual programming authoring environment (**Figure 8.2**, left) and a 3D modeling workspace (**Figure 8.2**, right). After authoring the behaviours, the system guides the users through simple steps to modify the model as dictated by the behaviours. To aid in the visual programming process, designers can test behaviours in real-time on the mobile device independently of the 3D model.

8.3.1 VISUAL PROGRAMMING ENVIRONMENT

Designers author the behaviour of their smart object using a high-level node-based visual programming language interface. Inputs can be placed on the canvas and linked to outputs to enable the creation of basic behaviours. The mobile device, which runs the Pineal mobile client, is always checking the state of the visual programming canvas, and accordingly updates and allows for live testing. The visual programming language is composed of the following components.

Input Modules

Pineal provides support for several inputs commonly found on mobile devices, including discrete inputs (those that have an explicit ‘trigger’) and continuous inputs (those that respond to changing, always-available values).

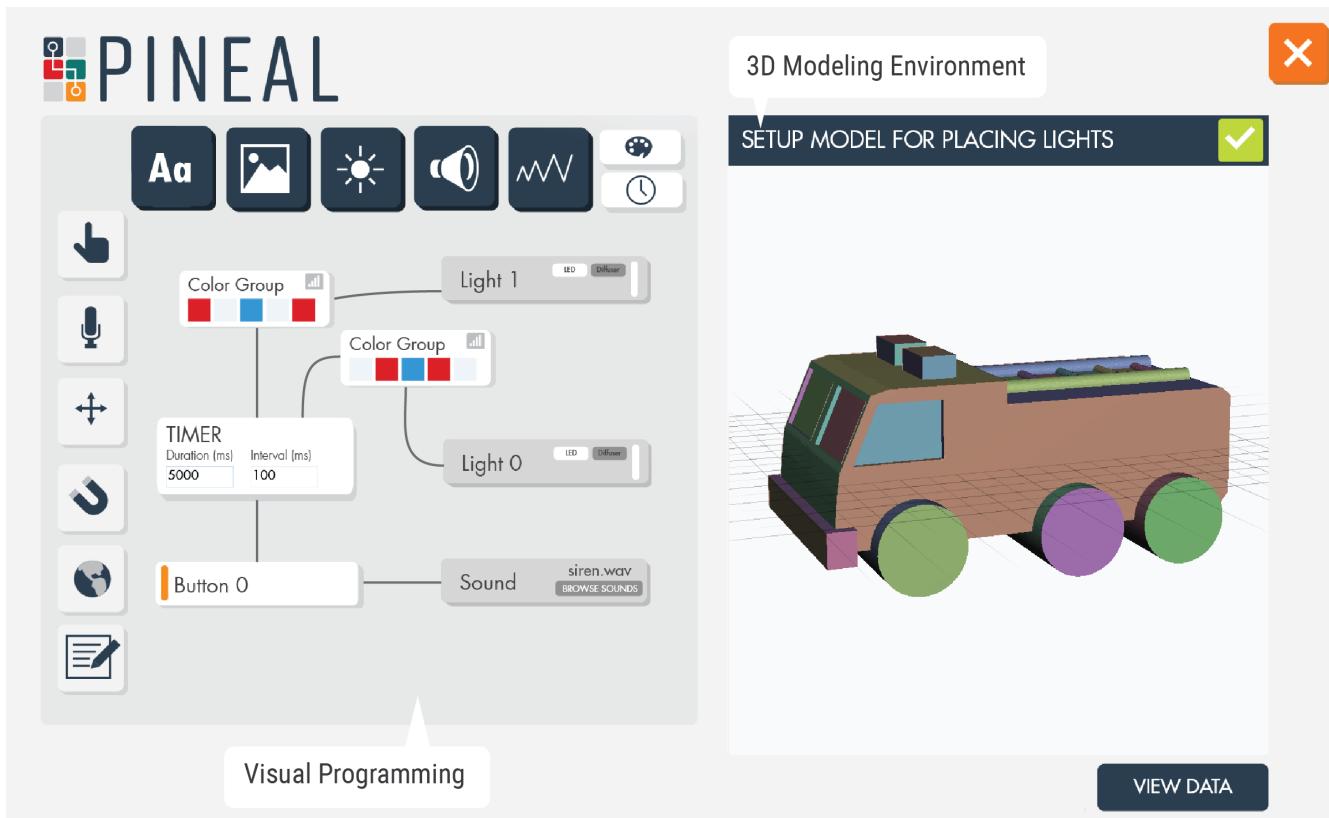


Figure 8.2. Pineal consists of (1) a simplified visual programming language to author basic behaviours for mobile devices (left); and (2) a 3D modeling environment, which allows designers to import custom models that can be 3D printed (right). Pineal's programmed behaviours then automatically modify the 3D models so they can indeed house the mobile device and expose the necessary inputs and outputs once 3D printed. This figure shows a Toy Firetruck model, which is used as a running example, where pressing a button makes the truck flash its lights and play a siren sound.

Button (discrete). Triggers an event when the generated button is pressed by remapping the user's touch directly to the touch screen via conductive material. Buttons can be of arbitrary shapes, and must be printed in conductive plastic.

Speech (discrete). Triggers an event when the specified word is sensed. Speech recognition is done by the mobile device, with events being sent to the server once the word is detected. Multiple speech modules can be placed onto the canvas, and thus the system can detect and respond to various words.

Shake (discrete). Triggers an event when a device is shaken. To recognize shakes, Pineal processes the accelerometer data and triggers the event when the magnitude of the acceleration exceeds a pre-defined threshold.

Orientation (discrete & continuous). Triggers an event when the specified axis of orientation (azimuth, pitch or roll) exceeds a given value, or reads a continuous value that can be mapped to another module.

Web (discrete & continuous). Triggers an event when a specified web event occurs (e.g. a hashtag is tweeted about), or provides a continuous value corresponding to the temperature forecast for a given city. Currently, only these two web functionalities are supported, but this feature could be extended to other services. The web service is selected by a dropdown menu.

Output Modules

A number of types of output are supported by Pineal to allow the smart objects to have expressive characteristics.

Text Display. Displays a text sequence when an action takes place. This sequence is fetched from a specified sequence every time the module is activated. The module also shows a set of sequencing options to iterate through the list, in incremental or decremental order, or in a random fashion.

Image. Displays an image selected from the pre-loaded library of images. The image location (x and y) can be updated via the input. The input is assumed to be screen coordinates, and appropriate mapping must be done by using the appropriate module sending the input.

Simulated LEDs (small light points). Simulates the effect of an LED turning on when triggered by routing light from the screen to another location on the object via light pipes. The module takes a list of colours as input, and each time the module is triggered it reads for the next element in the circular sequence.

Light Diffusers (large light sources with 3D form). Changes the colour of the display pixels when triggered. This enables the smart object to function as a low-resolution ambient display. Conceptually, this is similar to the simulated LED module, except it allows the user to import another 3D model to be placed on top of the base model as a light rather than having the light reroute to appear as an LED.

Sound. Plays a wave formatted file selected from the pre-loaded library when triggered.

Mapping Modules

Mapping modules support the input and output modules by providing means to store values and invoke timers. Currently Pineal supports two mapping modules:

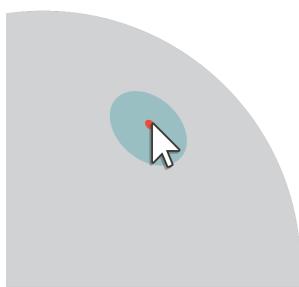
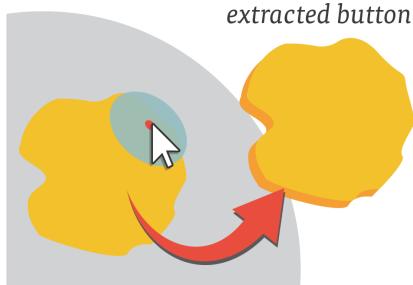
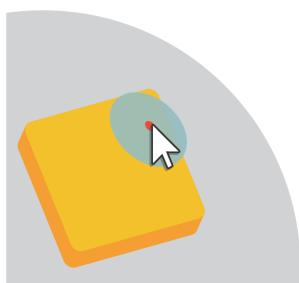
Retrievable Sequences. Text sequences and colour groups are lists that contain multiple entries. A calling module (e.g. input module) retrieves a value, which can be provided in increasing order, decreasing order, or random order, as specified by the user.

Timer. Fetches a retrievable value from a sequence for a specified duration at a given interval in milliseconds. A discrete input value can start a timer.

8.3.2 MODELING ENVIRONMENT INTERACTION

As the behaviours are put together, the modeling environment loads a set of instructions to modify the 3D model and creates a new form suited to the mobile device. In this section, I describe the types of interactions that the user can partake when interacting with the 3D modeling environment. The specific implementation details are later discussed in §8.5. While Pineal does a lot of automation to modify the 3D model, the system relies on an interplay with the designer to ensure they can customize the model to the way they want it to look and feel like once it is 3D printed (e.g. allowing them to consider where to place the screen, buttons, etc.).

The user can explore the 3D model within the environment (e.g. panning and zooming within the workspace). The Pineal interface displays a series of steps that the designer can take to continue modifying the 3D model, following the pattern of a traditional user interface Step by Step Wizard (described in Chapter 4, §4.3.8). As a step is loaded, the instruction to the designer is shown on the interface on top of the 3D modeling workspace (**Figure 8.2-left**). Once the designer agrees to step through the modeling tasks (by clicking the green checkbox to engage in that step), the system will automatically select the appropriate tool to enable the designer to customize the model (e.g. to place the screen of a device, the model of the screen is dropped and can be dragged across the surface of the base imported

A HOVER**B BRUSH SELECTION****C OBJECT PLACEMENT**

3D model). In addition to the standard panning (right click and drag) and zooming (scroll wheel), there are three types of interactions designers can perform, where the current tool or modality is determined by the current instruction. These interactions are defined by the Autodesk MeshMixer interface.

Hover

Designers can hover on top of 3D models (**Figure 8.3-A**), which shows the cursor as well as additional feedback (a circle projected onto the 3D model mesh coordinates) to suggest the point one is targeting.

Brush Selection

To place a button (**Figure 8.3-B**), the user “*paints*” a selection on the surface of the 3D model. Alternatively, double clicking will select the immediately surrounding “face” of the mesh. The system will extract that portion of the model, and extrude it upwards, creating the model for the button.

Object Placement

Screen/Light Placement. A model of the mobile device screen is placed atop the surface of the existing model (**Figure 8.3-C**), where the designer can drag and drop the model of the screen around and it will always be perpendicular to the mesh. The screen model then is translated towards the centre of the model and defines the splitting plane. In the case of a light, the system places a small sphere that acts as an anchor point for the light’s end-point.

Model Placement. Users are prompted to import a new 3D model (e.g. one which will become a light diffuser) which gets inserted into the

Figure 8.3. Interactions in the 3D modeling environment are of three types: (a) hovering the cursor shows where an operation will take place; (b) pressing down the left button and dragging allows the user to paint a selection; and (c) placing an imported model. Operations are always relative to the surface of the 3D model mesh.

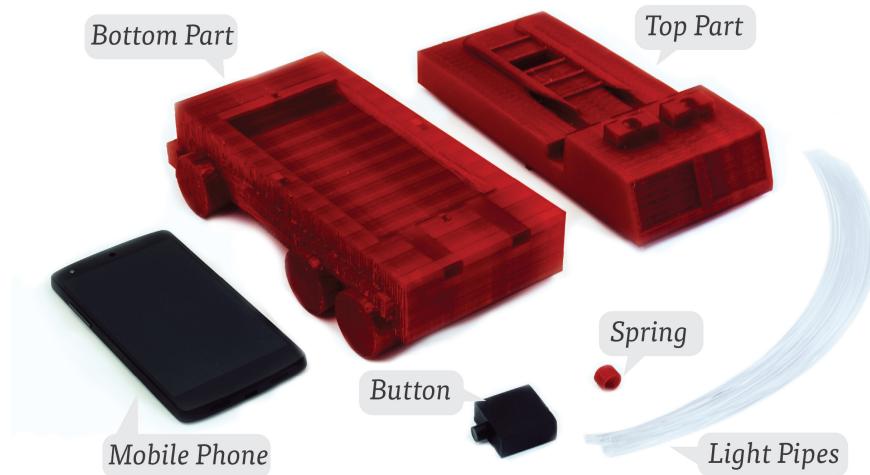


Figure 8.4. Components of the complete firetruck model.

modeling environment and incorporated as part of the larger model. The designer then can move the imported model around the original model via drag and drop. The drag and drop operation, like the screen placement, moves the imported model's base always relative to the surface of the original model.

8.4 USAGE SCENARIO: CREATING A TOY FIRE-TRUCK

To show how a potential user might work with Pineal, I illustrate a walkthrough of the various steps a designer takes to go to create a functional prototype. This scenario demonstrates a prototype with some degree of sophistication given that it uses a variety of inputs and outputs, and serves to highlight Pineal's threshold (i.e. how easy it is to get started). In particular, Pineal brings the focus of the authoring to the trigger-action interactive behaviours. The authored behaviours then serve as instructions to generate the appropriate form, as reflected in **Figure 8.2**. Note that the specific implementation details

of how the behaviours and the form modification take place are explained after the scenario, in section §8.5.

Pre-Conditions. To accomplish any prototype, designers must have: (1) a desktop computer to work with Pineal; (2) a mobile device to test the programmed behaviours and act as the prototype’s “*soul*”; and (3) a means to realize the prototype’s “*body*”, such as a 3D printer (to fabricate Pineal’s generated 3D model).

The Scenario. A designer creates an interactive firetruck toy where pressing a button plays a siren sound and triggers two flashing lights which are illuminated in alternating red-and-white colours. As a result, this scenario exemplifies a smart object with one physical input, and three different outputs, which lead to a form with multiple modifications, resulting in the physical components shown in **Figure 8.4**.

8.4.1 STEP 1: IMPORTING THE BASE FORM 3D MODEL

Prior to interacting with Pineal, the designer acquires a 3D model of a firetruck they wish to make interactive and imports it into Pineal. This model can be downloaded from common online repositories such as Thingiverse.com, or can be created using other tools (e.g. using Autodesk Fusion 360 or MeshMixer). The designer also selects the type of mobile device they are working with (phone or watch), which defines the dimensions of the mobile device for future mathematical operations.

8.4.2 STEP 2: AUTHORIZING THE BEHAVIOUR VIA VISUAL PROGRAMMING

After importing the model, the designer begins to create the behaviour of the device using the visual programming interface. The visual

programming components can be selected from a palette of inputs and outputs and arranged in the central canvas within the programming environment. The program flow for the Firetruck is visually represented as a schematic in **Figure 8.5**. Every time the visual program is modified, the new instructions are sent to the mobile device live: the system adjusts what inputs to listen for, interprets them based on the visual program, and prompts the mobile device to provide the appropriate outputs accordingly. As a result, the designer can try out and modify the program continuously. To create the program for the firetruck, the designer first selects the '*Button*' input from the palette and drags it onto the canvas. Then, they drag the '*Sound*' output from the toolbar onto the canvas and select a siren-sound from Pineal's library of sounds and connect the sound output to the button press. To create the lights, the designer wishes to alternate between different colour patterns. The designer drags two '*Light*' modules, along with corresponding '*Colour*' modules onto the canvas. The designer then specifies the lists of colours of the two colour modules, showing different arrangements of white, red and blue, which are set to sequence linearly. To make the lights change colour, the designer connects the button to a timer, which runs for 5000 milliseconds and updates every 100 milliseconds. As a result, whenever the button is pressed, the timer starts. The timer is then connected to the colour groups, and each colour group is connected to the corresponding light. Each time the timer updates, the next colour in the list is sent to the corresponding light. Once the sequences

run out of colours in the list, they go back to the beginning and continue pushing a new colour every 100 milliseconds until reaching the 5000-millisecond duration.

8.4.3 STEP 3: GUIDING THE MODELLING PROCESS

Once the user has defined a behavioural description of the object, Pineal guides them through the process of placing the physical components within the virtual model. The system runs through the visual program to determine the necessary manipulations the designer can make. While the generalized order of modelling steps is described in §8.5.2, **Figure 8.6** shows a schematic of the interplay between the designer and the system to generate the desired 3D model for the firetruck prototype. Given the set of instructions, Pineal automatically splits the firetruck into two pieces about the centre of the model, and creates a cavity for the phone along on the bottom half. The system also generates four alignment pins so that the two parts can be assembled together later on: four posts in the top half, and four holes in the

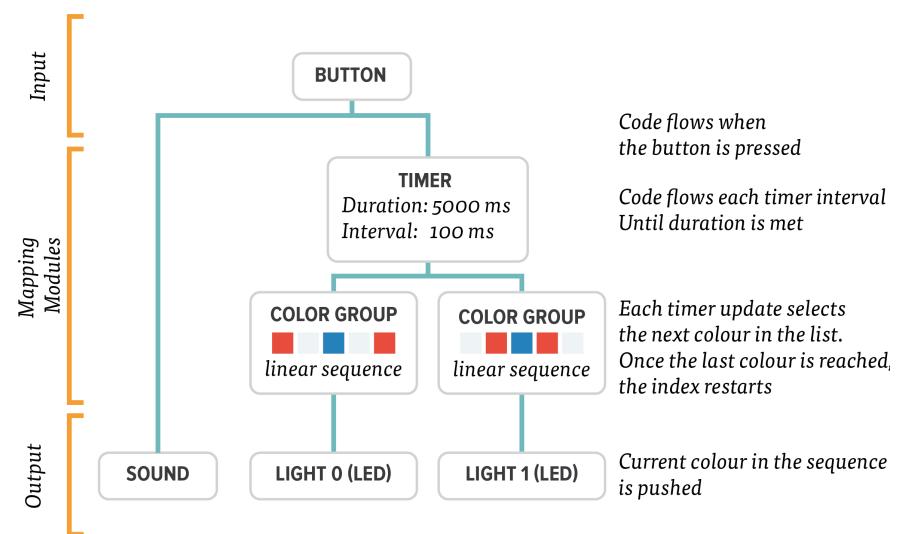


Figure 8.5. Schematic of visual program to create interactive firetruck.

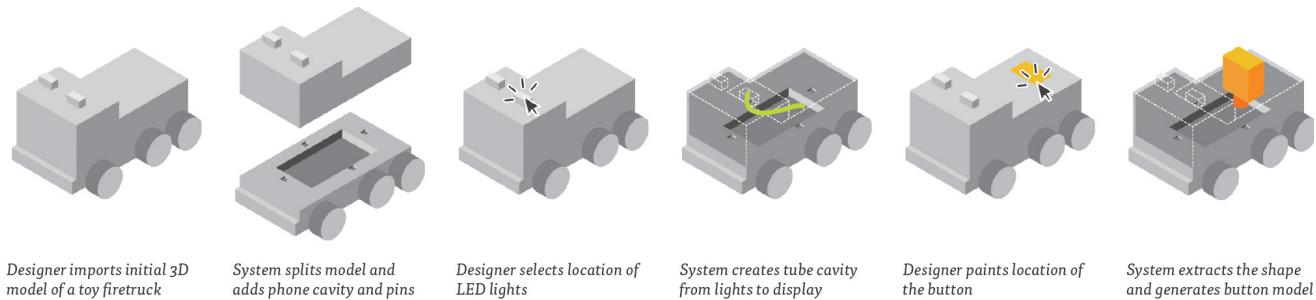


Figure 8.6. Steps taken by the designer to customize the firetruck 3D model from the moment it is imported until all modifications have taken place. The figure also shows the operation that the system performs in the background.

bottom half. Pineal then instructs the user to specify the location of the two lights, which the designer does by directly clicking on the surface of the 3D model within the 3D modelling workspace. Following this, Pineal automatically creates a curved tubular path from the surface of the model, through the object, terminating at the screen of the mobile device. Each half of the mobile device will display the colour for each light. The generated tubular path provides a channel where the designer will insert light pipes that will redirect light from the screen's pixels to the desired location. Next, the designer adds the button by selecting and painting the desired region on the surface of the model. This selection defines the shape and location of the button. Finally, since the user will be using audio output, Pineal creates an array of '*speaker holes*' through the object to better allow sound to travel out of the device.

8.4.4 STEP 4: OBJECT GENERATION

Once the designer completes their walkthrough, Pineal automatically generates the following objects that the designer can export: (1) the top of the fire truck (with alignment pin posts, a button cavity, and two hollowed channels for the lights); (2) the bottom of the fire truck (with a phone cavity, holes for the speakers, and alignment pin holes);

(3) a button; and (4) a spring. The spring 3D model is automatically generated, which can be attached to the bottom of the button to provide a more realistic button feel if desired.

8.4.5 STEP 5: OBJECT ASSEMBLY

The top and bottom of the firetruck can be printed in any material, while the button requires conductive material for the phone's touchscreen to detect when it is pressed (e.g. using conductive PLA). The optional *spring* is printed in an elastic material (e.g. NinjaFlex). The spring can wrap around the button to require activation force so that the button to trigger a contact event. After printing, the designer can begin assembly. Assembly is relatively quick and simple (final pieces shown in **Figure 8.4**), with this example taking approximately 5 minutes to assemble. The phone is placed in the cavity, and the two halves of the model snap together. The alignment pins, as well as implicit affordances (e.g. overall shape) help guide the assembly. Next, the light pipes can be inserted into the channels carved out for them, and the button can be placed into the corresponding hole in the

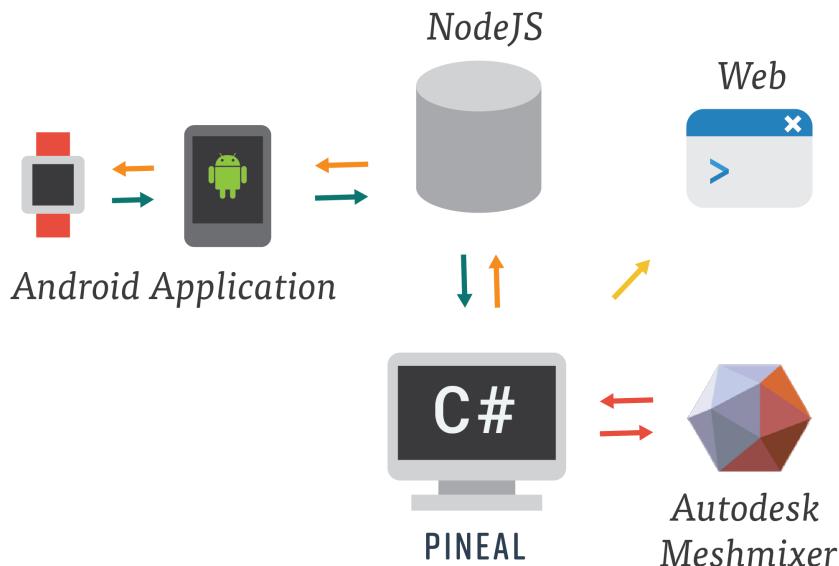


Figure 8.7. System architecture of Pineal, showing connections between smart phone, watch, NodeJS Server, C# client and Meshmixer

back of the truck. The firetruck is now ready for use, and when the button is pressed, it will play a siren sound and flash its lights.

8.5 IMPLEMENTATION

Pineal is comprised of a visual programming language, allowing designers to author high-level behaviours using a drag and drop interface. An interactive 3D modelling canvas updates as the user modifies these high-level behaviours. When changes in the visual programming language require modifications to the model, the interface displays the required actions above the modelling environment to guide the user through the process.

8.5.1 SYSTEM OVERVIEW

Software Components. The Pineal system (Figure 8.7) includes a smartwatch, smartphone and desktop PC, each running custom software. Currently, only the Google Nexus 5 phone and a Samsung Gear



Figure 8.8. Visual description of window transparency: the 3D modeling environment in Pineal is an instance of Meshmixer that sits behind the Pineal window.

Live smartwatch are fully supported and both run a custom application implemented in Java for Android (SDK 23), as they were the development devices available. The smartphone application connects to a NodeJS relay server on the desktop PC, which in turn connects to the Pineal C# (WPF) desktop application featuring the visual programming and modelling environments. This C# application connects to an instance of Autodesk Meshmixer, using the Meshmixer API³. This API allows Pineal to perform all the necessary automated modelling steps.

To realize the implementation of the watch application, a workaround needed to be devised. Given that this particular watch (the Samsung Gear Live) does not feature WiFi connectivity, the only way to communicate between the watch and the desktop is by first reaching to the mobile phone via Bluetooth.

Deciding How to Deal with Inputs and Outputs. The application running on the mobile devices streams all sensor data to the C# desktop application, where the data is processed. The desktop application takes charge of interpreting when inputs from the mobile device took place by looking at the current input modules on the visual program. For example, if a shake module is added, the system checks the accelerometer data in real time, and triggers an event when a significant change in accelerometer data is detected. When inputs events take place, the application runs through all the visual programming pathways created that are connected to that input. If any output action is

³ <https://github.com/meshmixer/mm-api> – last accessed July, 2020

to be taken as dictated by the visual program (e.g., play a sound, display an image), the C# application sends a message back to the mobile device indicating what action is to be taken. This constant streaming of data through the central C# application via the relay server allows for live debugging and interactive re-programming of the smart objects. As a result, the designer can test the program as they write it without the need to compile the code.

Making the 3D Modeling Environment Part of the Application. Pineal does not actually have a native 3D modelling environment. Instead, it borrows an instance of MeshMixer on the desktop to create an interactive experience. Pineal leverages WPF's ability to work with transparency within the operating system, which means that gaps in the application that can be clicked-through. As a result, one can click through the window's hole and interact directly with the MeshMixer interface, as shown in **Figure 8.8**. Thus, the interaction with MeshMixer takes place in two ways. First, the system leverages the API to send instructions to MeshMixer so that it can modify its 3D models. Second, users can interact with the 3D modeling environment (e.g. pan and zoom), as well as manipulate the model itself once Pineal sends the model manipulation instructions through the API (e.g. placing the LED lights, adding a screen, painting the button).

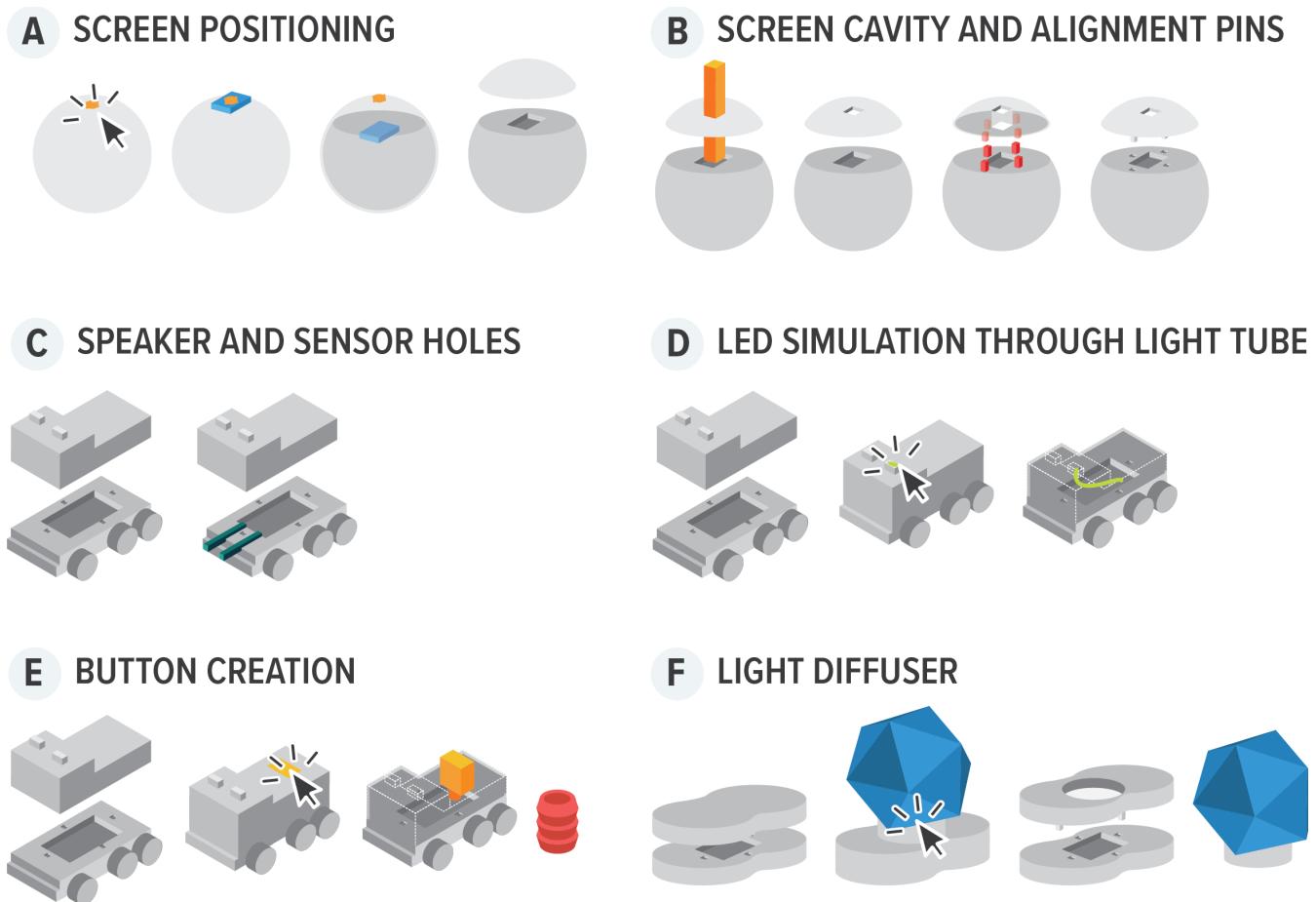


Figure 8.9. Operations performed by Pineal on the 3D models, which include (A) positioning the screens; (B) carving the screen and placing alignment pins; (C) Speaker and other sensor holes; (D) Buttons; (E) Simulated LED lights through light pipes and (F) Light Diffuser.

8.5.2 AUTOMATED MODEL CONFIGURATION

Pineal steps the user through different actions that modify the 3D model to fit the mobile device and any new modifications (e.g. buttons). The types of operation vary depending on the authored behaviours, and are summarized visually in **Figure 8.9**. To generate the forms, the system relies on applying a series of Boolean operations⁴

⁴ Boolean operations are a set of instructions to combine two polygons into a new shape. One can add two shapes together (union), subtract them, or intersect them.

that get applied to the user-imported base-model (e.g. the firetruck). This process is not fully automated, as the user has the flexibility to provide input for locating the inputs or outputs.

Defining the Mobile Device Screen Position on the Form and Placing the Mobile Device

By default, the device is placed in the centre of the 3D model. Images, text, and light diffuser modules require the display to be exposed, and thus are configured by the user in the modelling workspace. As shown in **Figure 8.9-A** once the user places the screen on the model, the device model is moved 2 cm towards the centre of the base model. This distance is the result of tests I conducted with multiple 3D prints to ensure the screen is visible while still being securely fastened within the object. If there are no instructions to place the screen in a custom position, the system places the device model inside the base 3D model's centre, facing upwards and parallel to the ground. The top surface of the device model acts as a cutting plane. The system then cuts a plane on the base model, creating two pieces: **top** and **bottom**. The device model is subtracted from the bottom piece of the cut to create a cavity.

Screen Cavity and Alignment Pins

As shown on **Figure 8.9-B**, Pineal can create openings on the device, as well as alignment pins. This is done through Boolean operations between the base model and new models (invisible to the user) generated at runtime: 30 x 30 x 50 mm for the watch screen (measurements that fit most smartwatches without a strap), 0.9 x 0.9 x 11 mm for the alignment pins. The system aligns the watch screen carving object with the screen centre of the device model, and then subtracts it from the top piece of the base model.

Alignment pins work in a similar fashion. Each alignment pin is duplicated to create one post and one hole for each alignment point. The hole is scaled at a rate of 1.1 in every dimension for clearance, so that the new printed object will fit together. Finally, the alignment objects are centered on the division plane in different locations with respect to the mobile device model (3 mm from the edges of the device cavity). The system performs a Boolean union between the top model and the posts, and a Boolean subtract between the bottom model and the holes. The number of alignment pins can be changed programmatically within Pineal's code from 4 up to 8. However, 4 seemed to be enough for generating stable attachments. The pins are placed parallel to each edge about the centre of the segment, and to place more than 4 the new pins are placed about the top and bottom of the edges.

Speaker and Sensor Holes

In the same way that Pineal supports alignment pins and screen carvings, it also supports carvings for other sensors (**Figure 8.9-C**). Currently supported configurations include speakers and camera, but one can also imagine creating holes for the microphone, volume buttons, etc. which can be easily integrated into the current system. In the current implementation, the phone volume description includes a set of tags that define the relative size and position of the camera and speakers. Indeed, for these holes to be generalized one would need to tag the models for mobile devices to define where the sensors are (if applicable). This can be done with built-in functionality of Meshmixer, by adding and naming *pivot points* by hand onto a device model.

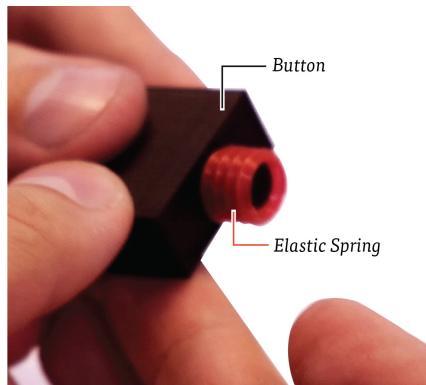


Figure 8.10. Example of the optional spring around the button as done in the Firetruck prototype. The spring is soft and elastic and requires the user to apply an activation force for the button to make contact with the mobile device screen.

Button Creation

Pineal can accommodate modifiers that transduce or reroute input (defined in Chapter 6 §6.4), such as buttons, as shown in **Figure 8.9-D**. To place a button, the user paints a brush selection in a region above the display location (since it requires capacitive input). This selection is smoothed by the system and then is extracted as a flat, separate object. This object is then extended downward to meet the screen, and is duplicated. The duplicate is subtracted from the top piece of the base model, while the original, now the new button, is rescaled by a factor of 0.9 for clearance. The system finds the centroid for the button face touching the screen, and adds a 7.5 x 7.5 x 5 mm cylinder. The button model can then be printed using a conductive material (in our case 1.75 mm Proto Pasta PLA). The conductive PLA proved a reliable trigger when contacting with human skin. To create restoring force, Pineal creates an optional spring model to be printed with a flexible material such as Thermoplastic Elastomer (in our case, NinjaFlex). The spring model wraps around the button's cylinder, as shown in **Figure 8.10**.

LEDs Simulated with Light Pipes

Another modifier is achieved using light pipes (fiber optic cables). The current prototypes use 1.5 mm diameter fiber optic cables from Industrial Fiber Optics. From different trials conducted, I found these to have the best light transfer and variety of available thicknesses as many cables can be bundled together. The light transfer is dependent on the screen brightness, with newer and higher-end phone models emitting much brighter lights. While the light pipes are

only 1.5 mm wide, the system generates 5 mm tubes, so that the user can fit multiple cables within one opening.

To add the lights, the user selects a location on the model, as shown in **Figure 8.9-E**. The Meshmixer client then creates a tube from that location to a screen coordinate of the mobile device. The system attempts to have the lights distant from each other to avoid the tubular cavities from crossing each other. To achieve this, the system will favour the corners of the screen. The mobile application then subdivides the screen into the number of lights placed, as determined by the program logic. For example, if two lights are placed, the screen will be divided in half. Each region of the mobile device will change colour as defined in the visual programming instructions.

Diffusers

As shown in **Figure 8.9-F**, a user can import models to create custom shaped ambient lights that are illuminated by the mobile device's screen. These ambient light models attach on top of the base model that have already been cut by the system. The user can import a new model and place it on the base model within the 3d modelling workspace. The system then creates an opening so that the mobile device light can shine through it. The light structure model is hollowed to a depth of 1 mm, and then printed using clear material (in our case MakerBot Natural PLA).

8.5.3 RAW SENSOR VIEW

To aid in the debugging and understanding of the constructed smart objects, Pineal includes an interface which displays raw sensor values

that are live-streaming from the mobile device (**Figure 8.11**). Currently, Pineal provides raw views to acceleration, orientation and touch input data.

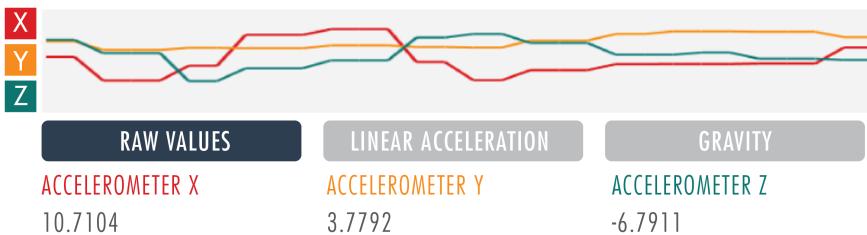


Figure 8.11. Live accelerometer data from the mobile device as visualized by Pineal. The visualization plots the raw X, Y and Z values.

8.6 RESULTING PROTOTYPES

To illustrate the breadth of use-cases and functionality supported with Pineal, I built five prototype objects. These sample objects are shown in **Figure 8.12**. The selection of these prototypes show first different levels of complexity and functionality, while also covering different aspects of the design space of the Soul-Body Prototyping Paradigm, as illustrated in **Table 8.1**. As a result, the prototypes demonstrate Pineal’s expressiveness by collectively covering a variety of inputs and outputs, and even web-based connectivity through novel and replicated examples.

8.6.1 TOY FIRE TRUCK

The sample walkthrough, described earlier, describes the workflow used to create the toy fire truck (**Figure 8.12-A**), which lights up and plays a sound when a tactile button is pressed. The firetruck is an example of using the mobile device to enable visual and audio output. It

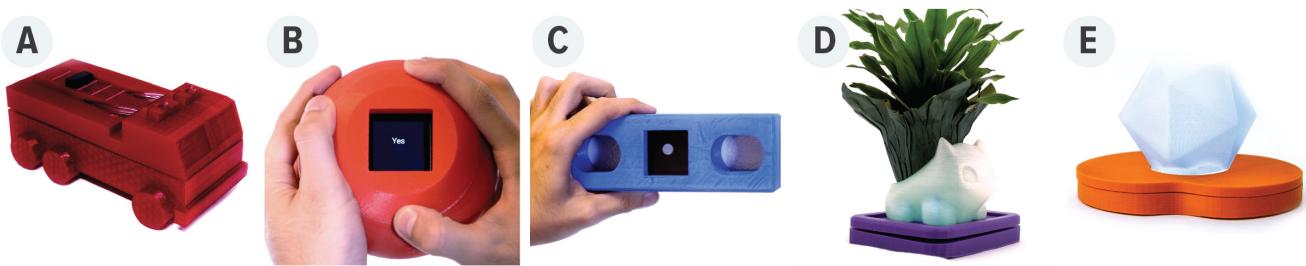


Figure 8.12. Sample of smart interactive objects created with Pineal, which includes: (A) Toy Firetruck; (B) Magic 8 Ball; (C) Level; (D) Ambient Light Planter; and (E) Voice-Activated Light Bulb

is also an example of transducing input – translating the phone's capacitive sensing into a physical button that provides tactile feedback when pressed.

8.6.2 MAGIC 8 BALL

A Magic 8 Ball (**Figure 8.12-B**) is a ball with a display that reveals a random answer to a question when shaken. To construct this smart object, a designer first creates a model of a sphere and loads it into Pineal, which will adapt the model to a smartwatch. Once the sphere model is loaded the designer uses the visual programming language to create a module to sense when the object is shaken, and sets an output to display one of the following strings randomly: ‘yes’; ‘no’; ‘maybe’; ‘try again’; ‘never’. The modeling steps can then be carried out, in this case consisting of specifying the location for the display and then clicking ‘generate model’. The model is automatically generated and can be exported for printing. The print creates two halves, with alignment pins, that house the watch. When the ball is shaken, a new response appears on screen. After testing the device, the designer can add a new response by editing the text field within the output module to include ‘wrong question’. The logic updates in real-time. This example shows the use of discrete motion input triggers and visual output.

8.6.3 LEVEL

A level (**Figure 8.12-C**) provides visual feedback to indicate when a surface is parallel with the ground. Currently there are existing mobile apps to simulate the functionality of a level, however, the form factor of a phone is not well-suited to being used as a level as it will easily fall over. To create a level in Pineal, a designer imports a model for a level, to which they add the smartwatch. The visual programming interface is used to map the horizontal location of a bubble image to be proportional to the watch's sensed orientation. The designer can then select the location of the screen on the model and export it for printing. This prototype demonstrates how Pineal is able to create linear mappings between inputs and outputs, and shows how motion-based input and visual image-based output can be utilized.

8.6.4 AMBIENT DISPLAY PLANTER

An ambient display (**Figure 8.12-D**) changes its colour in response to data – in this case, live Twitter data. To develop an ambient display with Pineal, the designer first imports a base model (a rounded box). They can then use the visual programming language to follow the Twitter hashtag `#CHI2017`. They add a diffuser module and import another model for the light diffuser, a Bulbasaur planter. Finally, they add a colour group module and populate it with a list of colours, connect the Twitter module to the colour group module, and connect the colour group module to the diffuser module. Each time a new tweet

	Firetruck	8 Ball	Lightbulb	Planter	Level
MOBILE INPUT					
CAPACITIVE					
ACOUSTIC					
MOTION					
VISION					
MOBILE OUTPUT					
VISUAL					
AUDIO					
VIBRATION					
CONNECTIVITY AND POWER					
CONNECTIVITY					
POWER					
REROUTING					
TRANSDUCING					

Table 8.1. Soul-Body Prototyping design space as fulfilled by Pineal. Highlighted areas in green show which dimensions of the design space were explored by each prototype.

is detected, the light scrolls through to the next colour from the list, thus actively changing colour. This means that if there are many colour changes happening in a short amount of time, there is a high amount of activity online. Pineal also supports other live web data

sources such as weather information. This ambient planter replicates an existing commercial product, the Ambient Orb⁵.

8.6.5 VOICE-ACTIVATED LIGHT-BULB

A voice controlled light bulb (**Figure 8.12-E**) could add ambience to a room, allowing the user to change the colour of the light by speaking to it. To create a smart lightbulb that responds to voice commands, the designer creates different speech modules with different words: “off”, “yellow”, “blue”, “red”, “white”, “green”. Each of these is respectively mapped to a different colour group containing an individual item: black, yellow, blue, red, white, and green. All of these colour groups attach to the light diffuser module. Now, each time a word is recognized, Pineal will send the appropriate colour to the light command, to which the phone will respond by changing the screen’s colour. The designer then modifies the model by importing a lightbulb model and placing it on top of a round base model which houses the phone.

8.7 DISCUSSION

The purpose of Pineal was to operationalize Soul-Body Prototyping. Consequently, the focus of the research was to explore the conceptual components of the system and implement them in a way such that the functionality supported the concepts. Given the technical complexities present in the system, there are limitations which I will discuss next. Despite such limitations, Pineal plays an important role

⁵ <https://ambientdevices.myshopify.com/products/stock-orb> – Accessed July, 2020.

in prototyping by allowing designers to focus the interactive behaviours, and also having a corresponding physical form as part of the prototype. Moreover, the lessons from the architecture implementation served to inform my next prototyping tool, Astral, which is described in the next chapter. In fact, I found that even after Pineal's completion, I still turned to it to generate forms for future systems, including Astral and early versions of WatchPen. I describe the limitations (§8.7.1) and discuss ways in which they can be addressed, or the potential compromise should the approach need adjusting. This leads to broader reflections on Pineal's role as a prototyping tool, its fit to current practices, and some of the decisions behind the approach (§8.7.2).

8.7.1 LIMITATIONS

There are two key types of limitations pertaining to Pineal. One is the limitations of the concepts and approaches taken when creating Pineal, and the other refers to the more technical implementation details.

Conceptual Limitations

Pineal's conceptual limitations refer to the reflection on what Pineal as a tool can and cannot do given how it was designed and what its purposes were. These result from aspects such as how the visual programming environment was realized, the lack of error checking, assumptions about the 3D modeling approach and scale in terms of the knowledge Pineal needs about 3D modeling geometries, as well as the variability among many mobile devices.

Creating a Generic Visual Programming Approach that can Inform Form Generation. The visual programming language was the result

of a three-step process. First, I ideated and generated sketches of potential prototypes using different types of sensors and outputs, and incorporated ideas and suggestions from collaborators. Next, we selected and prioritized a representative sample of the sketches that could explore a significant part of the Soul-Body Prototyping design space. Finally, I implemented the components for the visual programming as I generated the prototypes, making sure that (1) the previous prototypes still functioned, and (2) that it would be possible to achieve variations within the prototypes. The incremental approach is beneficial in that it ensures recombination of building blocks still works. For example, the Magic 8 Ball prototype was the first prototype developed and thus the shake and text modules were the first and only building blocks. When first designing and testing the colour module, I tested the system using the shake module to change colours of the screen whenever the phone was shaken. I then scaled the system so that it integrated speech recognition for the Voice-Activated lightbulb. At this point, I was able to test different combinations of shake and speech inputs to display colours or text. This process continued for each new prototype and set of modules that were being created. As a result, the system is capable of generating the prototypes presented in §5.6, as well as crossed variations resulting from manipulating the different building blocks. That said, the visual programming environment was a means to explore the concept rather than a fully fledged implementation. While it is possible to realize the prototypes shown in this chapter, as well as variations, there is no guarantee that more complex programs or specific variations will all function. Thus, what is possible is not exhaustive and is directly tied to the current example prototypes.

Focus on Trigger-Action Behaviours. The visual programming is limited in terms of the behaviours it can create. Pineal's design was inspired by other trigger-action systems, such as IFTTT⁶, which led to the expectation that following a standardized approach would cover a large range of interactive behaviours. As I created more of the prototypes, I realized some of the limitations of trigger-action behaviours, which led to the creation of the *Mapping Modules* to facilitate more continuous actions and responses. In particular, I became attuned to how different commercial applications or even games created rich experiences through nuanced behaviours. As a result, these lessons shed light at the opportunities for nuanced behaviours resulting from looking at the continuous data provided by multiple sensors. This led to further investigation on the meaning of interactive behaviour, and what is currently possible with authoring tools, which seem to be simplifying interaction design to such an extreme degree that flexibility has been compromised in favour of high standardization. When interfaces become highly standardized, there is less room for error, as there is more likelihood that the end-result will match the requirements. However, it becomes difficult to create something new, which again creates a need for specialization to allow the designer to work at a lower level and break the pattern of standardization. This is why there is a need for many prototyping tools that can fit different ways of thinking, and support a variety (of sometimes overlapping) results. These reflections became integral points in the

⁶ <https://ifttt.com/> – accessed January 2020

development of this dissertation, both the future systems and in further devising the theoretical foundations.

Lack of Error Checking. Because Pineal communicates with Meshmixer via its API, Pineal has no awareness of the geometry of the 3D model. Therefore, it will try and perform all operations (i.e. send mesh operations as commands) even if the model is unfit or will not work (e.g. it is too small to fit a mobile device). As a result, there is no way to know if the designer has chosen an appropriate model for the task. Furthermore, while the visual programming is a way to ensure the automation to modify the 3D model is done in an appropriate order, it does not guarantee that the end-result will work. Consequently, more complex models that leverage more than one placement operation (e.g. LED lights, exposing the display, adding a light diffuser) will likely not work as it would lead to conflict (e.g. resetting the splitting plane once a new instruction comes in). This limits the ceiling of the prototyping created with Pineal, but it also means that Pineal is not the appropriate tool for higher levels of complexity.

There is also no error checking for the visual programming, so nothing prevents a designer from erroneously linking two outputs together (something that should not be possible). Checking for erroneous inputs, or issues in geometries are important should the system move on to a sturdier implementation as a product, but does not affect Pineal's concept as a whole. That said, a potential avenue for future work is to explore ways for designers to better see the effects of their actions. For example, one could have an augmented reality model visible from a mobile phone camera or head mounted display that shows a preview of what a particular visual programming step

will do, how a mobile device might fit inside the model, as well as the effect of an input before the model is officially printed.

Geometry assumptions. It is important to note that Pineal makes implicit assumptions on the underlying 3D models. For example, Pineal assumes a 3D model is complete, meaning that there are no gaps, or self-intersecting triangles in the meshes. This can pose a problem when an object is for example, 3D scanned, and it will manifest itself in any 3D CAD environment. Next, it is important to distinguish the construction of the mesh in the 3D model. Meshmixer is also optimized for manipulating meshes rather via actions such as sculpting, addition, and subtraction of multiple 3D geometries, which is different from precise parametric design used to create professional industrial design models. The models that are best suited for Meshmixer should have a large number of triangles which are as evenly distributed as possible. This goes against the paradigm followed by many tools that favour optimization (e.g. Fusion 360 or OpenSCAD), as they optimize for the least number of triangles possible, and thus do not distribute the triangles evenly. When the meshes have few triangles and uneven distribution, mesh operations tend to fail. Meshmixer provides tools to solve this problem through operations that redistribute the triangles and increases the triangle density of a mesh. However, there is no simple way of telling what kind of mesh a 3D model has when browsing online, and thus the designer often would have to check and fix these models ahead of time. Meshmixer also provides additional features to optimize mesh operations, such as “*generating face groups*”, which looks at the overall collection of triangles and defines where the faces of the object are. As a result, hav-

ing a good understanding of 3D modeling and of a tool like Meshmixer greatly improves the likelihood of working with acceptable 3D models. In discussions with an industrial designer colleague, we discovered that the best suited models for Pineal are often those generated by tools that favour sculpting approaches, such as Cinema4D, ZBrush, Blender, or Maya. This is because sculpting-based tools have even distribution of many triangles as opposed to trying to optimize for the least number of triangles possible and perfect parametric design. These sculpting tools, however, are more tailored for character design for animation, where the aesthetic aspects come before the precise measurements of an object and their scalability. As a result, designers who want full customization and control when using Pineal need to understand how 3D models work and how different tools generate them. This level of understanding is not the same as becoming an expert with the tools like, say, an industrial designer. However, a lot of this understanding is required if one wishes to engage in activities such as 3D printing. If this becomes a deterrent, 3D printing may not necessarily be the best prototyping approach for a novice.

Pre-annotating mobile device models. The current implementation only has two models for mobile devices, a phone and a watch corresponding to the models used in the implementation. For these models, while they feature a simple geometry, the models needed to be annotated in Meshmixer to define the boundaries of the screen, as well as locations of the speakers, cameras, etc. Some of these aspects can be automated to some degree (e.g. one might define the screen as being a rectangle centered on one of the faces of the prism). To reduce the need for annotating models, one could generate a set of models that either (1) cover a wide range of possible devices that share

similar geometries; or (2) that designers can then modify to generate new ones. Alternatively, Pineal could register the dimensions of a mobile device, and ask about the locations of specific parts (e.g. volume buttons) as a step-by-step wizard, thus allowing the specification of mobile device models.

Mobile Device Variance and Limitations. Depending on the mobile device being used, the hardware limits the types of prototypes that are possible. For instance, the smart watch does not have rich audio output abilities, so designing a very small prototype that plays sounds would not work. This limitation will be addressed as devices gain richer capabilities and more devices are added to the system. In the future. Next, for very complex prototypes, not all desired functionality may be satisfied with a single device. For instance, if an object requires an LCD display on the top as well as the side, a second device would need to be added. This is not currently supported in the system but could make for future work exploring how to create more complex smart objects with multiple mobile devices.

Technical Limitations

Technical limitations refer to Pineal's specific implementation details, either in terms of what is and is not possible to build, as well as lessons learned later for future implementations.

Architecture Limitations. Looking back, one problem with Pineal's architecture was the constant "*daisy chaining*" of technologies. First, the smartwatch used in Pineal was limited in that it did not have wireless internet capabilities. This meant that the only way for the smartwatch application to run was to have the Bluetooth-paired mobile phone also running the application. Thus, communication with the

PC client from the watch would have to first make it to the phone and then to the relay server before it could reach the PC client. The relay server using NodeJS did not pose any major challenges at the time. In fact, the relay server from an architectural point of view, simplified much of the development. Given that there was more than one programming language involved in Pineal the relay server simplified the process since: (1) socket.io libraries are available for many platforms; and (2) the relay server creates a distributed model-view controller pattern, where the developer only worries about creating clients that share specific information. However, the disadvantage is that having a relay server creates an additional step when it comes to data transfer such as sensor data from the phone, as it effectively doubles any delay. The data first has to go through the relay server, which then goes to the destination, as opposed to clients communicating directly once. This proved to be an issue when working with more continuous and complex values in the following explorations such as in Astral, discussed in the next chapter, where I needed to transfer data such as live desktop screen captures. The next generation of implementations I created improved on the lessons from Pineal by (1) using a WiFi-enabled watch; (2) working with the same programming language whenever possible (C# Xamarin and WPF); and (3) reducing the number of connections, and while working at lower levels to achieve higher efficiency (e.g. traversing arrays, such as images with pointers).

Flexibility vs. Simplicity. Pineal's implementation remains close to the Soul-Body Prototyping paradigm. As a result, one can only rely on mobile sensors and outputs. This means that more complex func-

tions such as actuation or physical movement are not possible. Similarly, there are constraints from the prototyping materials themselves. For example, while it is possible to reroute touch points through conductive material, the material has to be conductive enough to work. In the case of the firetruck prototype, the conductive PLA print had to be of high density and it is likely that longer paths of rerouting would not work as reliably. One possibility for future work is to have custom electronics work in tandem with Soul-Body Prototyping. For example, .NET Gadgeteer or other microcontrollers could be integrated. However, the compromise is that the prototypes no longer become self-contained within the mobile device. The consequence is an added complexity both in terms of implementation as well as in terms of designer's assembly in a "*plug and play*" manner, as they would need to ensure multiple components are plugged in and running.

8.7.2 PINEAL, DESIGNERS AND PROTOTYPING

Reflecting on Pineal as a prototyping tool leads to many thoughts about the role of what designers would need so they can adopt a tool like Pineal; to what extent Pineal's prototyping is rapid; how permanent the resulting prototypes from Pineal should be; and to what extent tools should automate design processes.

Pineal, Expertise and Current Practices

Although Pineal simplifies the process of programming and form giving, now designers need to have more understanding of the underlying 3D models they are working with and the added constraints. Utilizing a tool like Pineal proposes a shift in mindset. First, Pineal forces designers to think about behaviours first, as the behaviours dictate

what the form modifications will be. Thus, the only way to think about form first, is to have a concrete idea in mind of what input or output is needed, and then create the behaviour for that particular feature. One could argue that the input and output modules, such as the button, or the text display are already reminiscing of the form. Still, it is different from an approach where the designer physically sculpts a form while programming interactive behaviours, where they can circulate back and forth much more freely. Should one remove the constraint of behaviours-first, however, automation would no longer be possible.

Is Automation Good?

Pineal performs some degree of automation when it comes to creating new physical forms. Computer science often values problem solving by generating a wide variety of alternatives through automatic generation. Generative design is an example of an extreme case. As described by Chakrabarti et al. (2011) generative design argues that computer scientists can create a grammar that “*computationally encode[s] knowledge about creating designs... which can be used to rapidly generate design alternatives*” (pp. 021003-2). Anderson et al. (2018) argue that systems as a result can generate “*a greater number of designs to be evaluated, and can enable the creation of designs that could not be possible by humans alone*” (pp. 3). Indeed, technology can look at certain constraints to create solutions that would be difficult for a human to generate, and even remove repetitive steps. For example, consider an example of *divergent generative design* by Matejka et al. (2018), where the goal of generating a monitor stand led to 16,800 alterna-

tives, which then were filtered to produce a set of 1,242 designs. Ultimately, choosing a particular solution is up to a person. And perhaps those 1,242 designs may not fit some of the qualities that person is looking for. There is no question about the value of generative design, nor a system's ability to generate solutions people cannot achieve easily (e.g. looking at specific and technical constraints). Yet, this means that (1) the systems that are automatically generating solutions for people require heavy engineering to ensure robust solutions; (2) that there is a need for a human-in-the-loop to ensure the solutions reflect the intention and process of the designer; and (3) that indeed, in the case of any blind spots in the generated solutions, the human needs to be savvy enough to be able to solve it. It is important to note that there can be a trade-off between automation and an individual's agency. On the one hand, having a system take care of the process can take away steps that are tedious, difficult or time consuming. On the other hand, systems should still enable users to craft where needed. With Pineal, it is possible to spend time crafting a form before it is imported into the system. The automation process takes away a lot of time-consuming steps, such as the creation of cavities for the mobile device, or creating alignment pins when models are split into two. But if the designer wants to carefully refine the model to fit a particular aesthetic, or they wish to nuance and fine-tune interactive behaviours to perform a particular animation, the automation might get in the way. In that sense it is important to identify Pineal as creating prototypes that can resemble final products, but by no means has the refinement or sophistication of a final product. This is where it is important to have techniques or other systems that can be used in tandem to continue evolving the prototype.

How Rapid is Pineal's Rapid Prototyping?

One term that often remains vague when discussing prototyping is how ‘rapid’ is rapid prototyping. It is possible to claim that Pineal indeed reduces the programming threshold by providing simple building blocks through its visual programming, and that Pineal indeed takes away the complexity of circuit building, or thinking about the form generation itself. This by default reduces the time dedicated to create functional prototypes. That said, the 3D printing process still requires some technical skill in operating a 3D printer, as well as assembling components. In addition, the 3D printing can take hours of idle time. For example, each half of the firetruck took 18 hours to print, which meant waiting overnight to see if the prototype fully worked. While this idle time might be considerably lower than creating a form from scratch, it is a time constraint that needs to be taken into account. In such cases there are two key elements to keep in mind. First, is that designers can use the idle time to work on another prototype or design activity altogether, so they are not completely stopped in the process. Second, is that the behaviours authored are immediately live on the mobile device. Thus, it is possible to have a reasonable idea of what the prototype will behave like, and one can use the 3D modeling environment preview to see what it will look like, so the assembly step at the end could be mostly a reality check. That said, if for some reason the 3D print is not successful or the model does not look or work as expected, it can be disappointing. This is simply a reality of these types of fabrication processes.

How Long Should Pineal Prototypes Live?

While the primary purpose of Pineal is to enable designers to rapidly prototype and iterate on ideas, it is possible that the created objects are of high-enough fidelity and resolution to serve as a permanent, functioning object. For instance, the Magic 8 Ball is of sufficient quality that it could be used as a customized novelty object for a child's birthday party, for instance. Future work is needed to explore what requirements vary when designing a '*body*' that is intended to permanently house a mobile device (e.g. considerations for charging cables or external batteries). That way, Pineal may encourage reuse and repurposing of old mobile devices to create new interesting objects.

Generality

Finding an audience today that operates with both smart object form design, as well as the design of interactions is difficult given how specialized these fields are. Perhaps industrial designers and makers today could directly benefit from a tool like Pineal, but the reality is that these are not common tasks they perform today. What remains beyond Pineal is the set of ideas that will go forward and inform the future of design tools, which can eventually help construct a new set of practices for designers. In the current implementation, the interactive prototypes along with the usage scenario evaluate Pineal's expressiveness in terms of the range of smart objects created (along with their possible variations), as well as its coverage of the design space of Soul–Body Prototyping.

8.8 CONCLUSION

Smart objects are ubiquitous, yet their design and prototyping requires substantial effort and knowledge in programming, circuit

building, and form-giving. Current mobile devices, such as smart watches and phones, possess a range of input and output capabilities that can be leveraged to prototype interactive devices. With Pineal, designers are able to rapidly prototype smart objects and modify both their form and function without substantial technical skills. Pineal uses the authored behaviours as means for automation to create new 3D models that fit Soul-Body Prototyping. The example prototype smart objects demonstrate a wide variety and use-cases that are enabled by this approach. Pineal can help designers overcome a large degree of designers' challenges of needing multiple expertise, lacking the necessary prototyping tools and needing close to product representations when prototyping. In realizing Pineal, it became possible to create interesting physical prototypes that could be held and tested. However, it also led to the discovery that interactive behaviour design could be much more than trigger-action behaviours, leading to the design and development of Astral.

CHAPTER 9. ASTRAL: BEHAVIOUR PROTOTYPING VIA FAMILIAR TOOLS

“Our hands feel things, and our hands manipulate things. Why aim for anything less than a dynamic medium that we can see, feel, and manipulate?” – Bret Victor

Recall my thesis statement: *we can repurpose existing hardware (mobile phones and watches) and software to enable designers to create live interactive prototypes for smart interactive objects.*

Thus far, I have shown in Chapter 6 how to repurpose mobile devices to prototype smart interactive objects, and in Chapter 8, I explored one way in which designers can repurpose a 3D modeling tool to generate forms to realize the Soul-Body Prototyping design metaphor.

The next step is to author interactive behaviours. As described in Chapter 6, this is challenging. Today’s design tools are modeled after desktop-computer paradigm and thus are limited to click-based transitions (e.g. tapping a button shows the next screen on a website). Even prototyping tools for mobile devices are still following the trend

of desktop-based design in spite of the added input and output possibilities. Yet, interactive behaviours are more varied and nuanced, as people perform actions beyond tapping (e.g. using fingers to pinch, flick or swipe, or tilting the device altogether). To transition beyond the current limitation of design tools, systems require more fine-grained real-time feedback that responds to the increasing richness of inputs. Doing so, people can better understand the effects of their actions and the system appears responsive and alive, as well as designed with care. To address the described challenge, this chapter presents Astral¹, a prototyping tool that allows interaction designers to author live interactive behaviours for mobile devices and smart interactive objects by repurposing existing and familiar desktop applications. More specifically, this chapter addresses the third research question posed in Chapter 1:

RQ3. How might designers leverage existing familiar software tools to author interactive behaviours for smart interactive objects?

I present this chapter in the following way. First, I discuss Astral in general terms (§9.1), and stipulate the related work and contributions (§9.2). Next, I describe different aspects of Astral's technical details, specifically the interface and the rule system (§9.3). To show how

¹ Portions of this chapter have been published in:

Ledo, D., Vermeulen, J., Carpendale, S., Greenberg, S., Oehlberg, L., & Boring, S. (2019). Astral: Prototyping Mobile and Smart Object Interactive Behaviours Using Familiar Applications. *Proceedings of the 2019 on Designing Interactive Systems Conference*, 711–724. doi: [10.1145/3322276.3322329](https://doi.org/10.1145/3322276.3322329)

Video figure: <http://davidledo.com/projects/project.html?astral>

Astral's components work, I present a usage scenario (§9.4). This scenario serves to demonstrate Astral's threshold and to some extent its ceiling, while also illustrating the nuances of how a designer might bring an interactive prototype to life. I also demonstrate how Astral can integrate and extend common prototyping activities (§9.5) such as the ones outlined in Chapter 4 (§4.4.1). In these scenarios, Astral converts the end-results of these activities into interactive prototypes working in the target device itself, such as a phone, a watch or a physical prototype that follows Soul-Body Prototyping. I finally, delve into the implementation details of the system (§9.6) before providing a larger discussion (§9.7).

9.1 ASTRAL

Astral is a prototyping tool for authoring interactive behaviours on mobile devices by repurposing existing desktop applications. The premise behind reusing desktop applications when interaction design has gone beyond the WIMP (Windows, Icons, Menus and Pointers) paradigm what might seem odd, yet the rationale is two-fold. First, desktop applications have the power and flexibility to author many aspects of interactive behaviour design (e.g. animations). Second, designers are familiar with existing desktop applications both as authors and users, where they can leverage tools such as web browsers, video editors or presentation software. To exploit these assumptions, Astral enables designers to display portions of the desktop screen on the mobile device, and provides building blocks for mobile device's rich inputs to interact with the desktop.

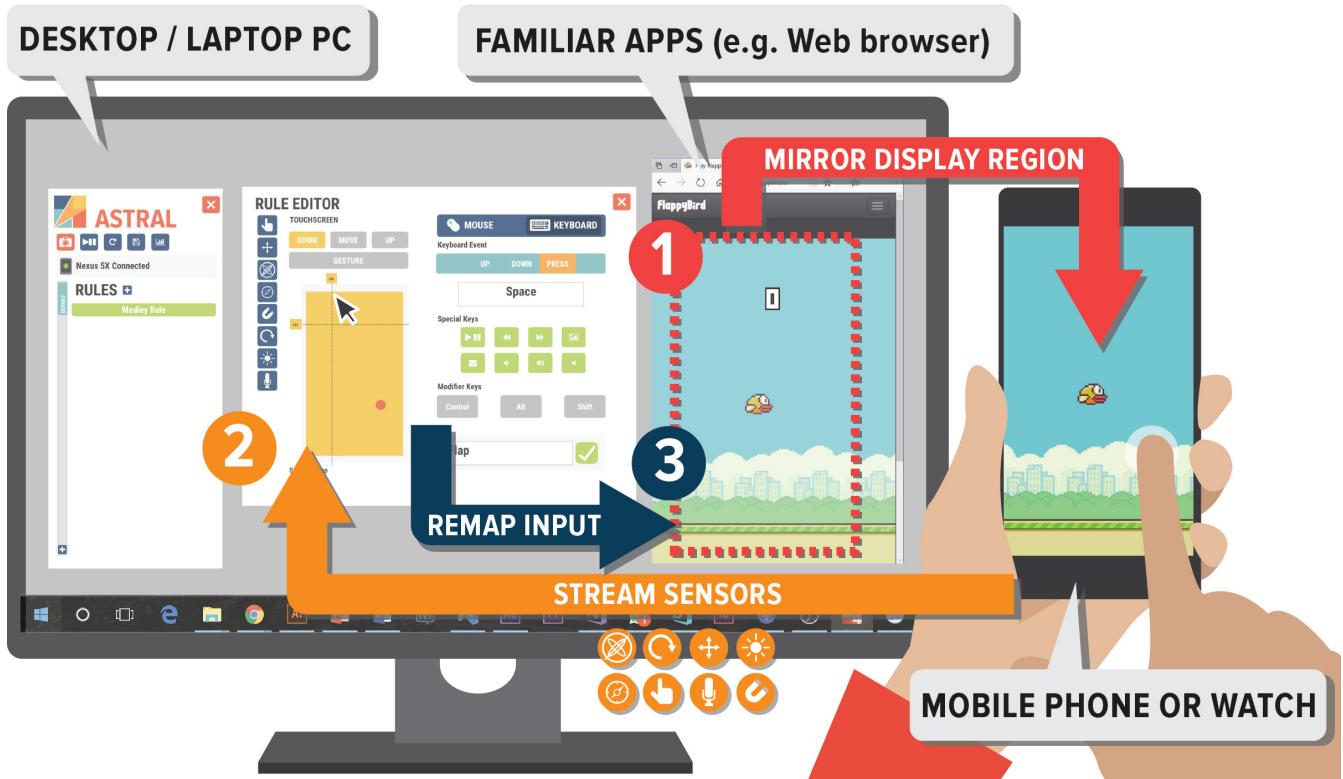
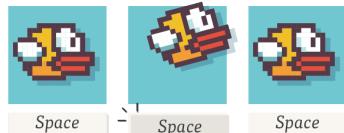


Figure 9.1. Astral allows designers to prototype interactive behaviours by (1) mirroring contents of a desktop region to a mobile device, (2) streaming mobile sensor data to the desktop, and (3) remapping the sensor data into desktop input (e.g. mouse and keyboard events) on a designer-chosen desktop application

Astral, as shown in **Figure 9.1**, works as two network-connected applications: a desktop and mobile client. The mobile client only has two functions: (1) receiving images from the desktop and displaying them live, and (2) sending the data of all sensors (i.e. potential inputs) of the mobile device to the desktop. On the other hand, the desktop application enables designers to explore and convert the sensor data to create commands that desktop can understand, therefore enabling manipulation of desktop contents via the mobile device. Thus, while end-users are technically operating the desktop computer in its entirety, Astral's features create the illusion that users are manipulating

ORIGINAL DESKTOP APPLICATION

Web browser showing FlappyBird.io: a game in which a bird flaps its wings when the player hits the space key to avoid obstacles



INTERACTIVE BEHAVIOUR PROTOTYPING ON MOBILE DEVICES WITH ASTRAL

1 DEFINE WHAT ONE SEES ON THE PHONE

By mirroring a portion of the desktop display, the phone shows a live copy of the monitor's contents. Phone's visual output is now exclusively a reflection of the desktop.



2 EXPLORE WHAT THE PHONE CAN DO

Through Astral's visualizations, one can see what phone sensors do as the phone streams live sensor data to the desktop.



3 MANIPULATE THE DESKTOP WITH THE PHONE

Astral converts the sensor data to desktop actions by simulating mouse and keyboard events. Here, tapping the phone sends a "space" key press. The desktop responds to the simulated key, the bird flies, and it is immediately reflected on the phone's mirrored screen.



Figure 9.2. Astral's steps to convert a desktop website into a mobile game.

and interacting with a mobile device. Given such approach, Astral belongs to the category of “*Smoke and Mirrors*” prototyping tools, described in Chapter 4 §4.4.3.

To realize the illusion that end-users are interacting with a mobile device when they are in fact controlling the desktop computer, Astral has three main functions: a portion of the desktop’s screen is constantly mirrored onto a mobile device (**Figure 9.1-1**), the mobile device streams its sensor data to the desktop in real time (**Figure 9.1-2**), and a set of building blocks enable converting the sensor data into mouse or keyboard events to remote control the desktop computer (**Figure 9.1-3**). These functions can be applied into a workflow of three steps, as illustrated in **Figure 9.2**: (1) defining what one sees on the phone, (2) exploring what the phone can do, and (3) manipulating the desktop with the phone.

Astral’s three workflow steps can be broken down using a deliberately simple example of converting a web game of Flappy Bird on the desktop into a mobile version. In the game Flappy Bird (<http://flappybird.io>), players can make a bird flap its wings by hitting the space-bar (**Figure 9.2**, top). The goal of the game is to keep the bird flying and avoid obstacles. Suppose a designer wishes to create a mobile version in which tapping the phone makes the bird fly. In this case the designer has to:

1. **Define What One Sees on the Phone.** As shown in **Figure 9.2-1**, the designer selects a region of the desktop display, such as a web browser running Flappy Bird. The contents of the desktop PC’s display are then mirrored to a connected mobile device (phone or watch) display in real time. Consequently, the mobile device shows a live

copy of the desktop monitor's contents, meaning that its visual output is exclusively a reflection of what is seen on the desktop. Any actions taken on the desktop, such as clicking or pressing a key will affect the desktop active application and therefore these changes will be seen on the mobile device. If the desktop monitor plays an animation or a video, the changing visuals will be immediately shown on the mobile device. In the example of Flappy Bird, the designer can see the main screen, or a live feed if the designer chooses to play the game on the desktop. Thus, at this stage, the mobile device only provides visual output.

2. Explore What the Phone Can Do. All the sensor data from the mobile device is streamed to the Astral desktop client (**Figure 9.2-2**). As a result, the designer can see custom visual representations (e.g. line charts) depicting the live data for multiple types of sensors, such as touch, acceleration, ambient light or the microphone. The designer can find the desired mobile sensors, select them and view their live data, as well as the individual parameters (e.g. the x-axis of the accelerometer). Through visual inspection of the data visualization, the designer can decide on a range of values of interest for that particular sensor (such as the touch-screen area, which shows the current touch point). While the designer can inspect the mobile inputs, actions on the mobile device do not have any effect on the desktop. Therefore, the mobile device at this point still only displays what is in view on the selected region of the desktop to mirror.

3. Manipulating the Desktop with the Phone. Now, Astral can enable the connected mobile device to control the desktop. This is done by

simulating mouse and keyboard commands which are executed virtually on the desktop's active application. Thus, the designer can convert the selected sensor data in the data visualization into mouse or keyboard events, which is achieved via rules. As shown in **Figure 9.2-3**, the designer converts a "tap" anywhere on the touchscreen into a spacebar keypress. Thus, every time the designer taps the phone, the Astral desktop client recognizes this action and executes a spacebar press and release. The active application, in this example the web browser with the Flappy Bird game, receives the keypress and makes the bird fly. Because the mobile device mirrors the selected portion of the display, the prototype appears to be brought to life on the mobile device: it successfully creates the illusion that tapping on the mobile phone makes the bird flap its wings and the response is seen on the mobile device screen. With very few changes, the designer instantly creates a temporary mobile version of a desktop game. Moreover, the designer can now quickly modify the set of rules to test different sensors. For instance, the spacebar command can be redefined/remapped as one blows on the microphone, or shakes the device (further exemplified in §9.5).

Because the mobile device shows a live view of the desktop, and the mobile device inputs affect the contents of the desktop, which are all reflected exclusively on the phone, Astral creates a *closed-loop of interaction* (of input and output) between the desktop and the mobile device. This loop of interaction can be seen in **Figure 9.1**, by following the flow of the 3 main operations of Astral. Systems to date only consider remapping inputs, or remapping outputs, without examining the interplay of both (see next section).

By creating a closed-loop functionality, Astral now allows designers to have a greater range of expressiveness: they can prototype and fine-tune interactive behaviours on a new variety of target devices (phones, watches and even tablets) and their variety of novel and interesting sensors (e.g. accelerometer, microphone, etc.). One can for example, repurpose a video editor so that a phone's side-to-side motion moves a mouse cursor across the video timeline on the desktop, while the video preview plays on the mobile device's screen at the same time. The result is a prototype where the speed and direction of the motion in the video are tied to the motion of the device. In this manner, Astral exploits existing desktop applications which designers already know and understand without the need to code. Moreover, Astral's use of mobile devices means that designers can apply Soul-Body Prototyping (Chapter 6) and extend the possibilities of behaviour authoring to smart objects. Overall, leveraging Astral and coming up with new ways of appropriating desktop applications opens the door to many new opportunities for interactive behaviour design and live testing of these behaviours on the target device itself, as opposed to a desktop simulation.

9.2 RELATED WORK AND CONTRIBUTIONS

Astral's design is informed directly by (1) prior formative studies (Maudet et al., 2017; Myers et al., 2008); (2) state of the art tools in research and industry; and (3) personal experiences. My personal experiences span talking with interaction and graphic designers, teaching interaction design to generalist designers and computer scientists, and creating prototyping tools and toolkits for interactive be-

haviour authoring in the past ten years. Based on the collected information, I have derived following four core design rationale decisions (R) for Astral, which corresponds to the system's contributions:

R1. Prototyping Live Interactive Behaviour on the Mobile Device.

Fast prototyping not only relies on expressiveness, but also on how quickly designers can preview and evaluate designs. Typically, there is a temporal gap between prototyping and testing. When constructing the interactive prototype, designers engage in some form of programming, as discussed in Chapter 4, §4.4.2, where the programming can take different forms such as arranging screen transitions. The program is then compiled which enables testing. The separation of authoring and testing forces end-users to constantly switch their focus of attention, and in consequence add difficulty to the process. Designer are forced to go back and forth between modifying the program and testing the prototype as part of the iterative design process. Hancock (2003) characterizes the distinction between regular and live programming as the difference between shooting arrows at a bullseye versus shooting water with a hose: the hose provides continuous feedback with which one can aim, adjust and shoot at the same time. I believe that by integrating the programming and testing, designers can always keep their goal in mind while making small adjustments to achieve the desired result. Astral acts like the hose in this case, and supports live authoring in two ways. First, designers can leverage the *closed loop of interaction* (§9.1) to simply run and execute a desktop application onto a new target mobile device with new means of manipulating the application (e.g. converting motion sensors to arrow key presses on a desktop's web map). Second, designers can run the

rules as they are being authored all while experiencing these behaviours on the target device itself (i.e. the mobile device). As a result, the designer can fine-tune and adjust the behaviours almost instantly (exceptions and workarounds are explained in the next section).

Astral builds on prior approaches to create the live prototyping experience. Victor's systems in *Inventing on Principle* (2012) show different strategies to achieve liveness in the context of coding, such as integrating the code and the program within the same screen, or showing the trails of movement history and allowing the developer to adjust the values while seeing that trail change given the new data. Different interface prototyping tools feature aspects that can help approximate live prototyping to help designers create interactive applications. In the area of mobile interaction, prior work has explored having a live custom UI builder mirroring the desktop screen's UI (Meskens et al., 2008), collecting and connecting photos of sketches which can be tested in an ad-hoc manner (de Sá et al. 2008), or demonstrating actions in one source device and replaying those actions into a new target device (Meskens et al., 2009). The demonstration-based approach is also seen in electronics programming. Exemplar (Hartmann et al., 2007) leverages programming-by-demonstration as a way to associate sensor patterns to actions, which can then trigger a key press on the desktop or control the mouse cursor.

R2. Providing an End-User Interface that Allows Designers to Explore Variations Among Mobile Sensors. Because the types of input data provided by sensors is non-trivial, one way to make sense of sensor information is through data visualizations. Such approach has been followed in prior systems such as A CAPella (Dey et al., 2004),

Exemplar (Hartmann et al., 2007), and MAGIC (Ashbrook and Starner, 2010). However, given the large number of sensors present on a mobile device nowadays, there are two new challenges. The first challenge is that the sensors on a mobile device are varied, and thus benefit from different visualization approaches based on the data provided. I address this challenge by including different visual representations according to the nature of the sensor (see §9.4.3). The second challenge is that performing a single action (e.g. shaking the device) triggers multiple sensors which makes it difficult to disambiguate. To solve this problem, I build an interface in which designers can record their actions on video (see §9.4.6). The video recording also shows a stack of visualizations of the sensor data values which the designer can inspect and select the right sensor. These two strategies help increase the expressive match² (Olsen, 2007), as one can choose between the live isolated visual representation or the video analysis approach depending on the task and goal. Moreover, custom visuals aim to look and feel more like what the designer is doing with the mobile device, such as showing a dial-like visualization for the compass, or a map of the screen for the touchscreen.

R3. Supporting the Use and Repurposing of Existing, Familiar Applications for Prototyping. When working on top of existing infrastructures, toolkits can leverage existing functionality to quickly explore new types of interactions. In itself, this is not a new idea. For

² Olsen (2007) defines expressive match as the “estimate of how close the means for expressing design choices are to the problem being solved”. For example, when selecting colour, a low expressive match is using hexadecimal code compared to a higher match through picking a colour with a colour picker.

example, Olsen (2007) already discusses how working with common infrastructures enables new technology combinations to support new solutions. he uses a scenario where pen input behaves as mouse input, and thus mouse-based applications can now be operated with a pen as the input device. Through screen mirroring and keyboard remapping Astral introduces the *closed loop of interaction*, which enables designers to use their own workflows and control *any* familiar desktop application to prototype novel mobile interactions.

While many prototyping tools in the research literature reappropriate native operating system functionality, the focus of the work is often only on one side of the equation, either focusing on inputs only or outputs only. For example, Exemplar (Hartmann et al., 2007) and MaKey MaKey (BM Collective & Shaw, 2012) support remapping sensor input into mouse and keyboard events, while Icon,(Dragicevic, 2004) which is a toolkit and editor, creates input-reconfigurable interactive applications. On the other hand, other systems have leveraged screen mirroring, as done in Semantic Snarfing (Myers et al., 2002), VNC³, and TeamViewer⁴, to display the desktop application's contents (or a portion of its visual contents) onto the mobile device. In the case of applications such as TeamViewer, the system provides a one-to-one mapping from touch input on a mobile device to a control a desktop's mouse cursor.

R4. Supporting interaction-driven animations. While desktop-based applications can often rely on trigger-action behaviours, they already

³ <https://www.realvnc.com/en/> – Last accessed April 2020

⁴ <https://www.teamviewer.com/en/> – Last accessed April 2020

feature behaviours that are hard to recreate in a way that does not require coding. For example, consider a single drag and drop operation, where one selects an item and can move it across the screen from one location to another. Such an action has an animation which is defined by more than just time, it is driven by the user's input. Few systems have operationalized this type of authoring without code through keyframing approaches (see Chapter 4 §4.3.6), such as Monet (Li & Landay, 2005) and Kitty (Kazi et al., 2017). With newer devices such as phones and watches, these types of interactions have the potential to become more common given the wide variety of rich sensors. Many operating system functions feature these types of highly nuanced and sophisticated animations, such as Slide to Unlock in iOS, Android's Quick Settings, or the animations that play while one emits a voice command to a digital assistant such as Google Voice or Siri. Astral can achieve *interaction-driven animations* on a variety of sensors thanks to fully-fledged desktop tools such as video editors thanks to the *closed-loop of interaction*.

Building on interaction–driven animations, designers must be able to see continuous live effects from their input, but moreover examine and modify *how* those effects take place. One way to address this '*how*' is through *easings*. Easing is a term used by Adobe Animate⁵ (formerly Adobe Flash) to refer to the slow-in and slow-out principle of animation (Thomas & Johnson, 1995). Here, the number of in-between frames are increased or decreased at keyframes between poses to create the illusion that an object is speeding up or slowing down.

⁵ <https://www.adobe.com/products/animate.html> accessed October 2018

Figure 9.3 provides examples of several different easing methods and how a circle would appear over time. Adobe Flash incorporated easings as a default linear inbetween (commonly referred to as “*tween*”) that could be applied to a change in motion (i.e. through position, scale, or rotation). Penner (2002) created scripts for Flash to change the character of the easing through mathematical functions, which further nuance the types of easings available. Easings, however, assume the animation plays as a function of time. Prior work has applied animations as a function of continuous sensor input, such as OctoPocus (Bau & MacKay, 2008). Other work applies Penner’s (2002) easing functions as a function of continuous inputs (e.g. Ledo et al. (2015), Kazi et al. (2017), and Reach and North (2017)).

In Astral, one can apply easing functions to continuous mouse-move events to fine-tune animations *as* continuous sensor-based interactions happen. This provides additional customization power to the authoring of interactive behaviours. The easing functions can produce aesthetic experiences, as well as more utilitarian functions such as balancing the sensitivity of an input’s effect. These continuous animations with easing functions are not explored in prior programming by demonstration approaches, as they tend to favour recognition of discrete events from continuous sensor inputs (e.g. Exemplar (Hartmann et al., 2007)).

To recapitulate, Astral extends previous approaches by combining existing techniques of mirroring, streaming and remapping to feed into new building blocks: the creation of small, self-contained rules that drive a lively and animated prototype. These rules allow repurposing familiar desktop applications in ways not seen before.

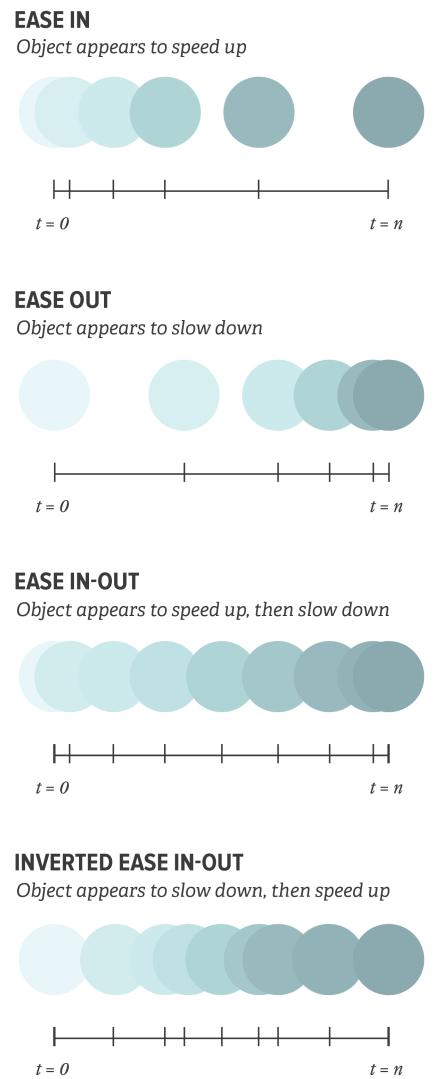


Figure 9.3. Easings in animation. A circle moves horizontally over time (from time = 0 to time = n). Ticks on the timeline mark the position of keyframes.

9.3 WORKING WITH ASTRAL

When using Astral, designers are working with their existing desktop applications, which carry the bulk of the work. Designers then can use Astral to make that prototype interactive on a phone or a watch. Therefore, it is meant to require minimal setup and leave as little trace as possible after its use so that designers can keep the source material the way they had it before. Astral's interface has few components to it (see **Figure 9.4**), which I describe next.

9.3.1 MIRRORING DESKTOP CONTENTS

Clicking on the camera icon (**Figure 9.4-A**), designers can mirror display contents onto the connected mobile device. Next, an overlay region, a rectangular window with the proportions of the mobile device, is shown. The window which can be moved, scaled, or rotated around the desktop screen. The area enclosed within the region contains the intended visual output for the mobile device. The pixels contained within the rectangular area are captured and mirrored live to the mobile device client.

9.3.2 SPECIFYING INPUT REMAPPING THROUGH RULES

Once desktop content is streamed to the mobile device, designers can author an interactive behaviour by defining a *rule*. A rule is a software abstraction that contains information as to how mobile sensor data is converted via simple mapping to keyboard and mouse events. This abstraction holds a source sensor type, a range of values to which the mobile sensor data is compared, and a destination mapping (mouse or keyboard event). To create a new rule, the designer clicks on the

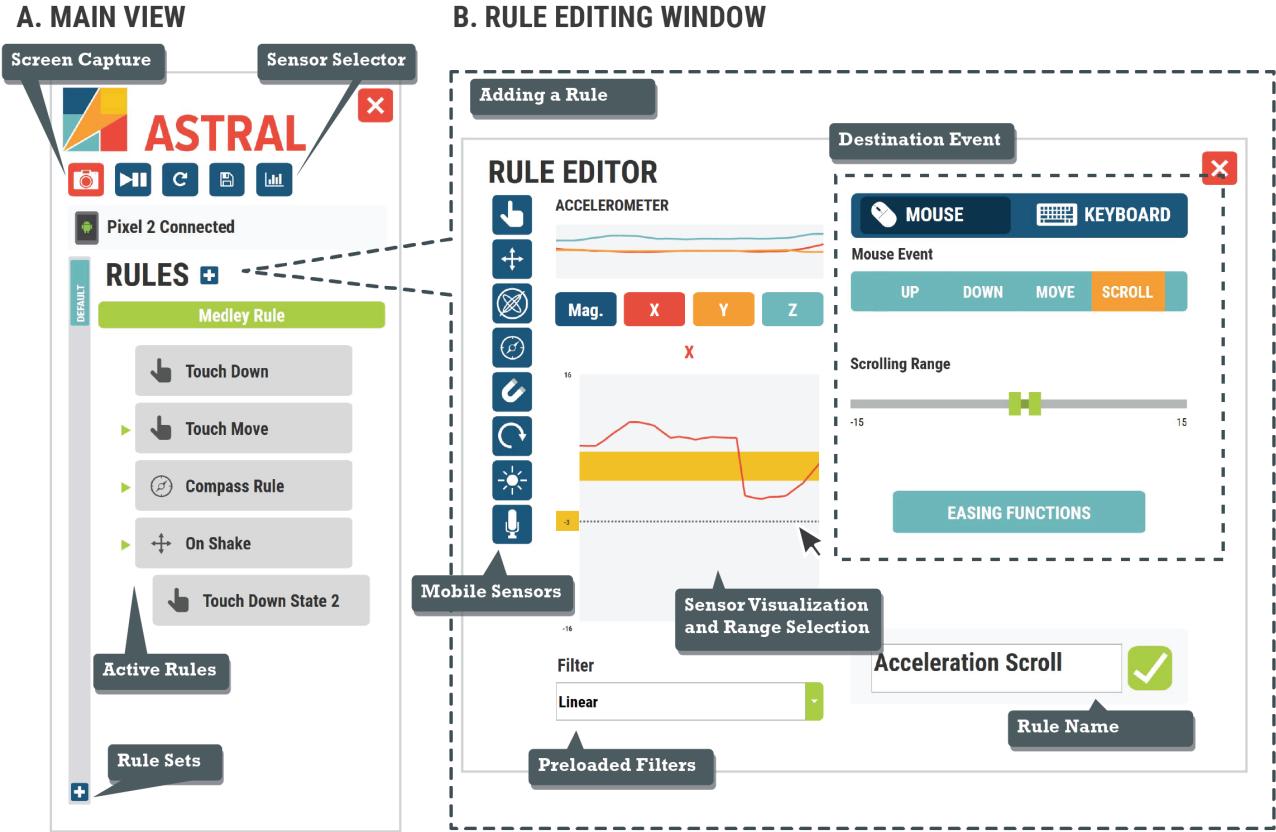


Figure 9.4. Astral’s main interface as displayed on a desktop computer, showing the (a) Main View, and (b) the Rule Editing Window, where designers can create mappings from sensor data onto mouse and keyboard events.

‘plus’ sign to open the *Rule Editor* (Figure 9.4-b) – a guided pop-up window. In the editor interface, a designer can select the source mobile device sensor, define the values of interest on the live visualization via direct manipulation, and assign a destination mouse or keyboard event. Figure 9.5 shows a schematic of a rule which will be used as a running example. The figure depicts the mobile device’s physical motion, the corresponding data visualization for the y-axis of the acceleration plotting the live response to the motion, and the resulting mouse move event which is the destination desktop input. To create this rule, the designer must use the Rule Editor, select a

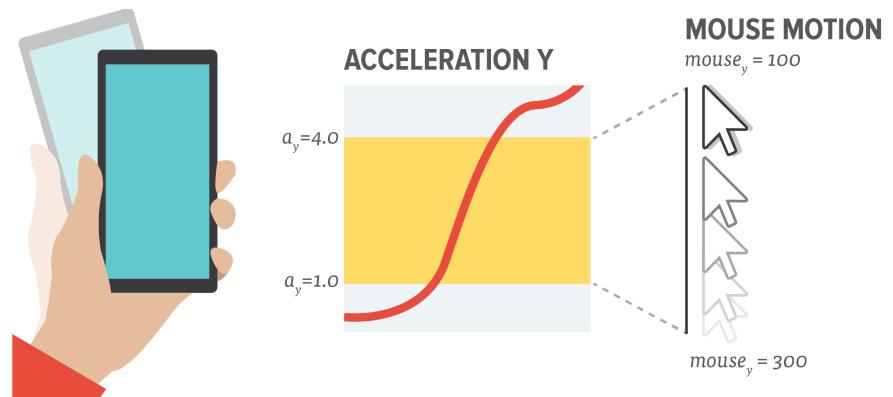


Figure 9.5. Rules in Astral take a sensor value, such as the accelerometer y-axis, and map it to a desktop input such as a mouse position.

sensor and its range of values, and create a mapping (i.e. convert the sensor data to a mouse or keyboard event).

Selecting a Sensor and Range of Values. The Rule Editor shows a list of sensors provided by the mobile device. A designer can choose the individual *sensor of interest* (Figure 9.4-B side panel) to define the rule. Clicking on a sensor icon reveals a live visualization of the sensor and its values to help the designer understand (1) the particular sensor's response as the device is being manipulated, and (2) whether the sensor is appropriate to use. The visualization is tailored to the selected sensor (and its parameters/individual data) to provide higher expressive match (Olsen, 2007). Examples of these custom visual representations are shown in Figure 9.6. In the example shown in Figure 9.5, the designer can select the accelerometer sensor from the Rule Editor side panel, and select which parameter to observe (the aggregate magnitude or x-, y- or z-dimension). In this case, the designer chooses the y-dimension. Moving the device forwards and backwards dynamically updates the visualization, where the red line shows the current value. They can use visual inspection to see the

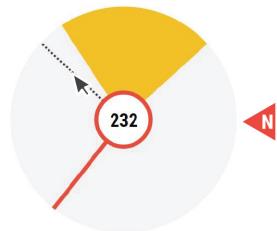
range of values of interest and then constrain the sensor values to a range such as 1.0 m/s^2 and 4.0 m/s^2 by clicking and dragging, which creates a vertical selection shown in yellow. Note that the visualization dynamically resizes as larger values are detected (e.g. if one were to shake the phone much harder, the acceleration ranges may increase significantly). Sensor readings can be further transformed by applying prepackaged filters (e.g. extracting gravity and linear acceleration values from the acceleration).

Converting the Mobile Sensor to Desktop Input via Mappings.

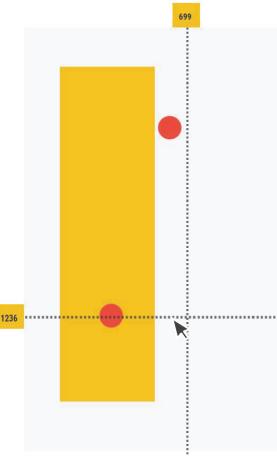
The designer can now map the mobile device sensor input to a *desktop input* (e.g. a mouse move event). Mouse events can be constrained on the desktop (as shown in **Figure 9.5**), for example, only allowing mouse movements vertically between 100 and 300 pixels. The rule editor offers the option to select the destination region for the mouse motion events through a rectangular selection region similar to mirroring overlay. Alternatively, one can define increments for mouse wheel events (Windows default scrolling: 120 pixels per step). For keyboard events, designers can specify a key event (key down, key press, or key up) and the associated key (e.g. arrow left, spacebar). Keys can either be typed or selected from a list of operating system defined keys (e.g. volume controls, media playback, print screen). One can even leverage shortcuts offered by the target application adding modifier keys (e.g. control, shift).

Astral implicitly distinguishes between *discrete* or *continuous* inputs, in line with Exemplar's categorization of sensor values (Hartmann et al., 2007). **Figure 9.7** shows how Astral maps a mobile input (in this case the accelerometer value) to a (a) discrete input destination (e.g.

A COMPASS



B TOUCHSCREEN



C LIGHT SENSOR

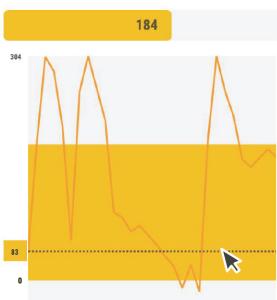
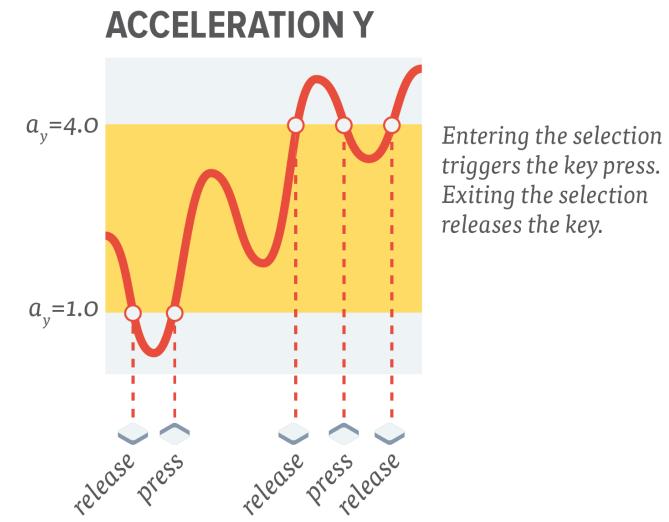


Figure 9.6. Sensor visualizations change depending on the currently selected sensor, to enable more straightforward mappings. The figure shows (A) Compass, (B) Touchscreen, and (C) Light Sensor.

A Mapping to Discrete Input Destination (Key Press)



B Mapping to Continuous Input Destination (Mouse Move)

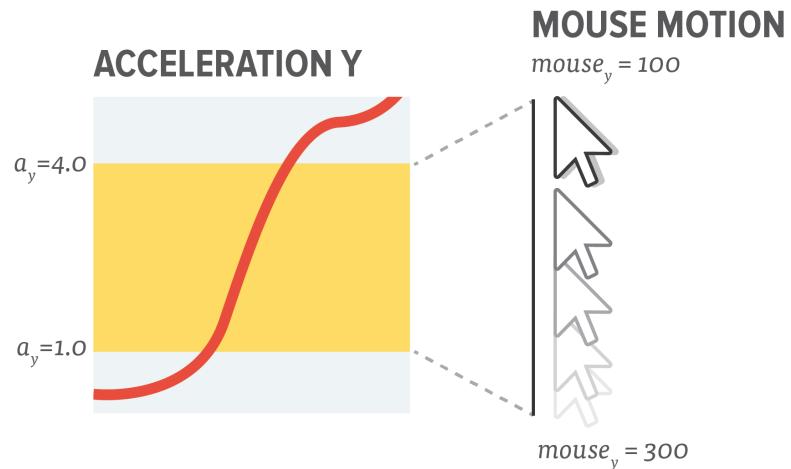


Figure 9.7. Mapping sensor inputs to discrete and continuous keyboard/mouse inputs. Figure shows (A) mapping to a discrete input such as a key press, and (B) mapping to a continuous input destination (e.g. mouse position).

key press) and (b) to a continuous event destination such as a mouse move event or a scroll action. When a discrete device sensor input (e.g. 1 or 0 for a proximity sensor) is mapped to a discrete desktop input (e.g. a mouse click), the inputs are mapped one-to-one (if one triggers, the other triggers). When a continuous device sensor input

(e.g. an accelerometer's x-dimension) is mapped to a discrete desktop input (e.g. a mouse click), it is triggered when the device sensor input enters or exits the designer's chosen range of values. When a continuous device sensor input is mapped to a continuous desktop input (e.g. a mouse move), the value is interpolated between the source range and the destination range (e.g. from the accelerometer's y-dimension to a mouse position within a selected range). Lastly, a discrete sensor mapped to a continuous desktop input will simply map to the two extremes of the destination values. Note that the system handles discrete and continuous values for both mobile device sensor (source) inputs and desktop (destination) inputs automatically based on the designer's mapping. This means that designers do not need to explicitly think about whether their source or destination inputs are discrete or continuous.

Easing Functions. Mapping continuous device sensor input to mouse motion or scrolling inputs allows designers to apply easing functions that interpolate between both (Penner, 2002). Instead of doing a linear interpolation, a designer can choose from a list of easings (**Figure 9.8**) which immediately change how the destination input behaves.

The authoring process is dynamic: designers can immediately view, test and modify rules as they author or edit them. If they want to stop the rule from running (e.g. because the mobile device input is taking over the mouse cursor), they can press the '*escape*' key to pause or play live mapping. This is particularly important to prevent **input locks**, where the mouse cursor is overtaken by the mobile device sensor during the authoring process. When the designer is finished, they

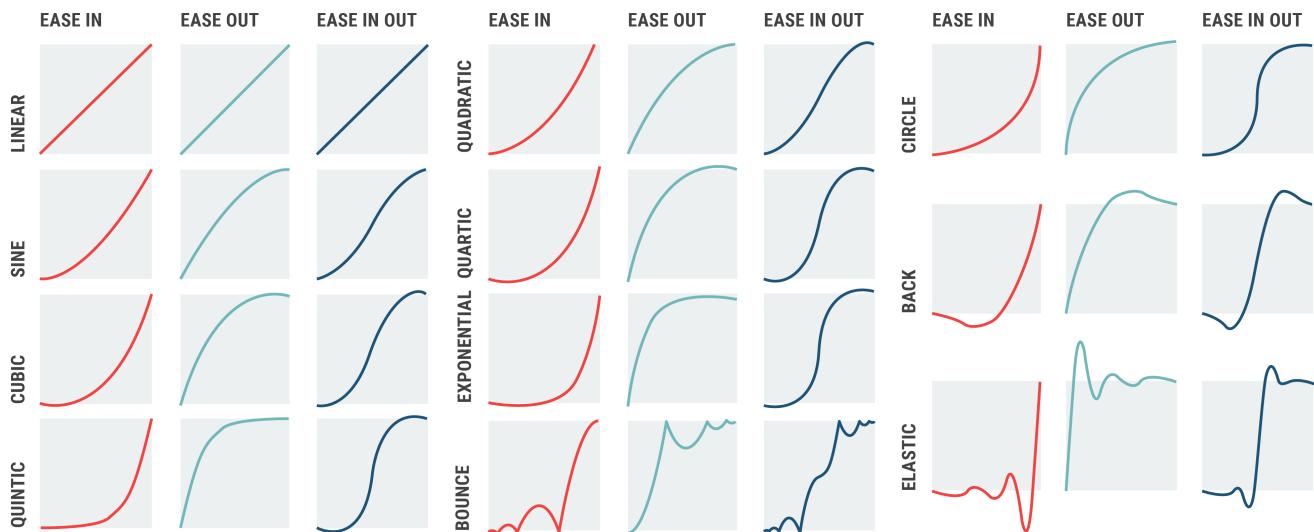


Figure 9.8. Easing function options in Astral based on Penner (2002). These easing functions can also be inverted.

can name the rule and finish creating it, which adds it to the active ruleset in the main application window.

9.3.3 MERGING SEVERAL RULES INTO RULESETS

A behaviour may often require several rules, potentially using different sensors. Astral adds an additional layer of abstraction, *rulesets*, to support combining rules. If a ruleset is active, rules within that set will execute as long as the mobile device streams sensor data.

To test variations of interactive behaviours, designers can create multiple rulesets and switch between them at any time. When there is an active ruleset, a newly created rule will be added to that set and stacked vertically along with the others.

9.3.4 DECIDING WHEN RULES ARE TRIGGERED

Astral aims for rules to be a minimal unit of mapping an input to an output. As a result, multiple rules can be combined into rulesets. However, Astral provides several additional structures that expand the expressiveness of a ruleset.

Conditional (When). When a device input meets a condition (e.g. values within a selected range), rules inside a conditional structure are activated. Conditional structures are always listening for input, and as long as the condition is met, all contained rules will execute. Thus, it is possible to implement techniques such as the clutch mechanism in *tilt-to-zoom*⁶ (Hinckley and Song, 2011) by nesting two conditionals. The first condition is *touch is down*, which holds true as long as a contact point is touching the screen. The second condition, which depends on the former one is that the touch is not moving. Thus, once the conditions of *touch is down* and the condition of *not touch move* (i.e. *touch is moving*, negated) are met, it is possible to interactively map *accelerometer Y-dimension* from the device to *mouse scroll up/down* desktop input. This example is elaborated in §9.5.2.

Sequence (Next). A sequence defines a chain of rule transitions, where different rules are chained in order, and once a rule has triggered, the next one becomes active. This means that designers can create interactions such as simple state transitions (e.g. where the mirrored portion of the display can shift, move or resize), or support rules that enforce an order. After a rule in a sequence is executed, it becomes inactive and the next rule in the chain becomes active. Each rule in a sequence can mirror different portions of the desktop screen. Through sequence structures, Astral can approximate state-based approaches (as done by design tools like d.tools (Hartmann et al.

⁶ Tilt-to-zoom is a one-handed mobile interaction technique. Users can pan across content by sliding their finger. When the touch is held, one can zoom in and out by tilting the device. The change in global state is referred to in HCI as “clutching”.

2006) and InVision⁷) without explicitly implementing states. That said, the chain is only a single linear pathway.

Medley. A medley switches the currently active ruleset to the next ruleset on the list when a device input rule is triggered. Designers can define one medley at a time. As a result, designers can use this special rule to quickly switch to and test different variations of a prototype (*getting the right design* (Buxton, 2007)). As a result, designers can sequentially test a set of prototype alternatives, thus being able to check different types of inputs and how they affect the overall experience. Astral can thus support experimentation with any variation within rules, including sensors, thresholds, easings, or desktop inputs.

9.3.5 SENSOR SELECTOR

Astral allows designers to disambiguate between multiple sensors. The Sensor Selector provides an overview of values from all available sensors as stacked line charts. Figure 9.9 shows the Sensor Selector interface, which shows a video on the top left corner (live feed of the desktop's web camera when recording, or the current frame of a recorded video once a recording has completed). By pressing the record button, the system records a webcam view that is synchronized with the different sensor data. All sensor data (except touch, which shows touch points over time) shows a corresponding line chart which updates its values as the recording takes place, displaying the different sensor values across time. Designers can scrub through the recorded video by hovering on the timeline or by hovering on the line chart

⁷ <https://www.invisionapp.com/> – Last accessed April, 2020

SENSOR SELECTOR

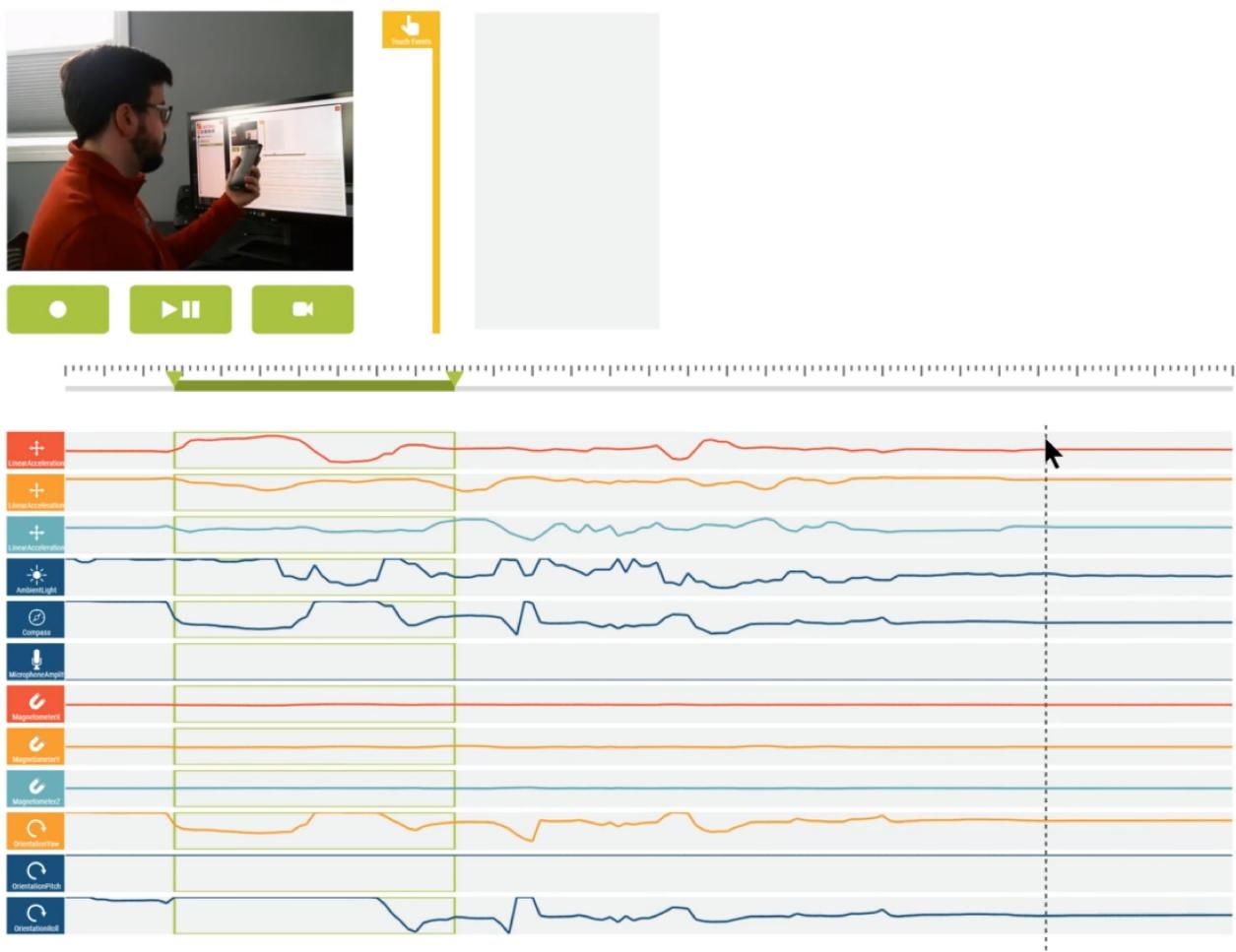


Figure 9.9. Sensor Selector. Designers can record and playback a video which plots the sensor data of all detected mobile device sensors. Designers can then scrub on the video or the visualizations to see the associated video frame to that sensor value. Designers can create a selection to automatically create a rule that uses the range of values for that sensor.

visualizations. Thus, designers can go through the video and inspect both video of the performed action and the visualization of the corresponding sensor data as stacked line charts. Hovering on the charts or the timeline updates the current video frame. The timeline features a range slider to select the area that reflects the designer's physical action of interest. From that selection, designers can see all sensors that reacted, and select a specific sensor to create a rule. The

system will open the Rule Editor with the sensor and its recorded ranges already filled in.

Having all sensors displayed together with the webcam view can help designers select the relevant sensor to use. This was inspired by prior experiences with students in computer science as well as discussions with designers, in which people were unsure as to what sensors corresponds to which particular actions. For example, it often came as a surprise that the accelerometer can detect the mobile device angle of rotation by looking at the value of gravity (9.8 m/s^2).

9.4 USAGE SCENARIO: CREATING A LEVEL

Having described Astral's interface and the nuances of screen mirroring and how to convert mobile sensor data to desktop events, I now describe a simple usage scenario to illustrate Astral's functionality. I showcase how a designer might work with Astral to create a lively prototype and author its interactive behaviours without any need for coding. For this example, the designer wishes to create a level (akin to a carpenter's level) on a phone. The level should portray a bubble that is centered on the screen when the phone is level, where that bubble moves to corresponding sides when the phone is not level. The designer can already use Adobe Illustrator and AfterEffects – familiar image and video applications (See §4.2) to draw and realize these visuals and nuanced behaviours. As no coding is involved, the designer can focus most of their time and effort on the aesthetic elements of the behaviours (e.g. animations, visuals) as opposed to implementation details. This scenario can also be seen in the video figure for Astral, and will be described below as a sequence of steps. I also note how long I spent in the first-time creation of the scenario,

which shows minimal time in getting the prototype to run⁸. **Figure 9.10** shows some of the steps in the scenario as a visual summary which includes a snapshot of the final result.

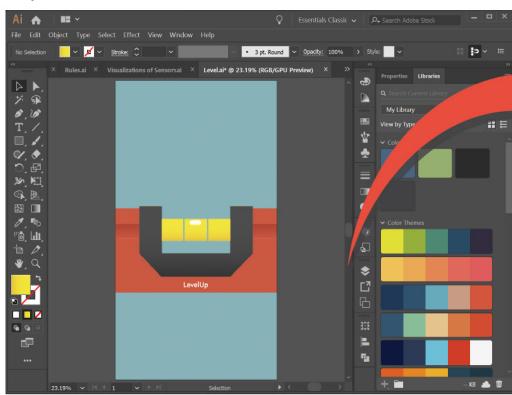
9.4.1 PREPARING THE PROTOTYPE: ILLUSTRATOR AND AFTEREFFECTS

This step occurs outside of Astral, where the designer is using familiar desktop tools. The designer uses Adobe Illustrator to create illustration of the level at reasonable fidelity. The “bubble” is extracted as a separate layer that can be masked and animated (15 minutes). The designer imports the Illustrator file into Adobe AfterEffects and creates a simple linear animation in which the bubble moves from one end of the level to the other as the video progresses through its timeline (7 minutes). The designer now has a video prototype on their desktop that communicates *what* happens, but not *how* it happens.

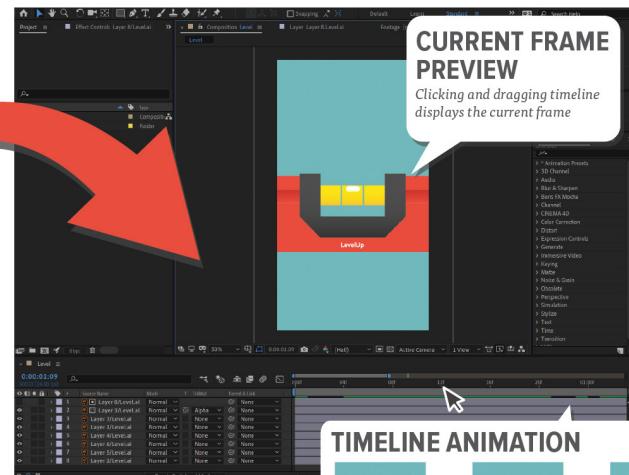
Normally, a designer would be stuck after this step, as they would be unable to transform this video into an interactive prototype running on the phone. Thus, it is not possible to truly test and interact with the prototype on the target device, and so it is difficult to achieve any additional sophistication acquired by fine-tuning the results. Astral can now bridge the gap, as the designer can transform this desktop video into an interactive prototype running on the mobile device.

⁸ Prior knowledge in Astral and the desktop tools influences design time. When creating this example, I was not very familiar with Adobe AfterEffects, but I consider myself an expert in using Adobe Illustrator and Astral.

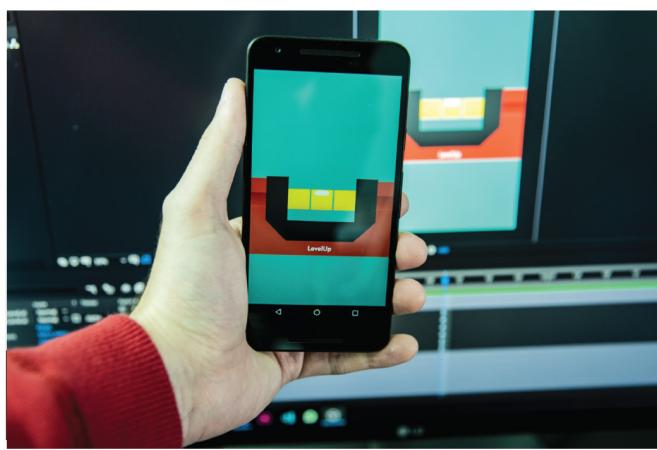
The designer creates Level in Illustrator and imports it into AfterEffects



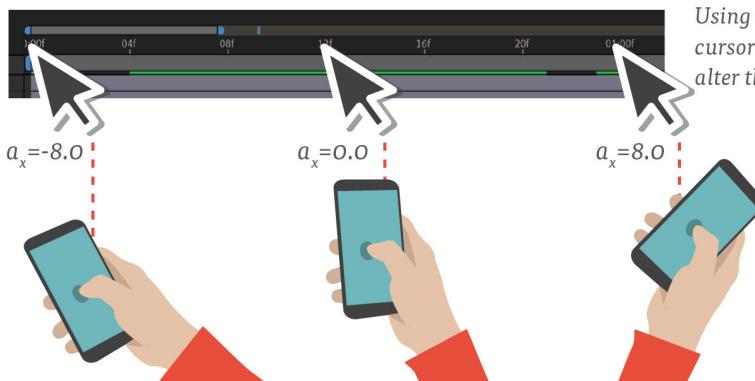
The designer animates Level in AfterEffects



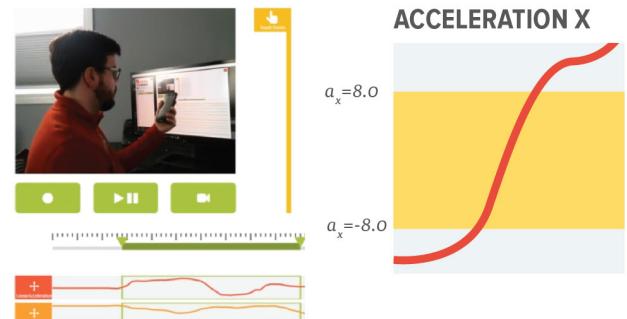
The designer mirrors portion of AfterEffects' current frame preview into mobile device using Astral



The designer adjusts the acceleration selection as he wiggles the phone. He then maps the mouse position to click and drag across the video timeline as a function of the x-axis of the phone's acceleration



Using the Sensor Selector, the designer realizes that side-to-side motions correspond to the accelerometer's x-axis, and creates a rule from it by right clicking on the Linear Acceleration X



Using easing functions, the designer can change how the mouse cursor behaves, thus affecting the live preview: the designer can alter the interaction-driven animations

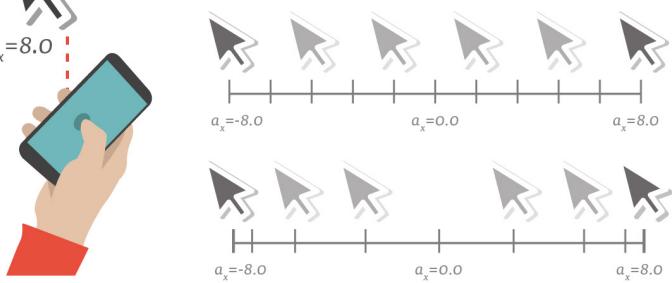


Figure 9.10. Visual description for Astral's usage scenario. In the scenario, a designer authors a prototype for an interactive level on a mobile device by repurposing the timeline on a video editing software.

9.4.2 STEP 1: STARTING ASTRAL

The designer launches Astral on the desktop (**Figure 9.10**) and connects the mobile device to it. The designer clicks the camera icon to select a region of the desktop to mirror onto the device. The region is the output video in the AfterEffects window, which will then appear live on the device (1 minute).

9.4.3 STEP 2: SENSOR SELECTOR

The designer wants the interaction to play out when tilting the phone from side to side in a portrait orientation. Unsure of which sensor might be used for this, the designer opens the Sensor Selector (**Figure 9.9**), and moves the webcam so that it can capture him in the act of tilting the phone. The designer presses record. Astral records all available sensor values, as well as webcam video of their actions, which shows him tilting the device from side to side along with other motions (2 minutes). The designer then plays back the video and sensor recording, to determine which sensor is being activated as the device motion takes place. From visual inspection and scrubbing through the video timeline, he finds that *Linear Acceleration X* and *Linear Acceleration Y* both react to the side-to-side tilt. However, *Linear Acceleration Y* is also triggered when the device is tilted forward and back and therefore is less valuable. Thus, the designer decides that *Linear Acceleration X* is the sensor of interest. He narrows down the area of interest by adjusting the range slider, right clicks on *Linear Acceleration X* and clicks on the “Create Rule” option, which opens the Rule Editor (1 minute).

9.4.4 STEP 3: RULE EDITOR

The Rule Editor (**Figure 9.4-b**) automatically selects *Linear Accelerometer X* as its active sensor parameter, and already has a defined range based on the readings from the Sensor Selector. The designer repeats the desired behaviour (side to side motion) and sees the values up close in the Rule Editor *Linear Accelerometer X* live visualization. He now adjusts the acceleration range on the data visualization via direct manipulation of the yellow-filled range (1 minute).

9.4.5 STEP 4: MAPPING MOUSE POSITION TO THE AFTEREFFECTS TIMELINE

The designer now uses input remapping to specify *how* the interaction takes place. He decides that moving the device from side to side should be converted to mouse click and drag actions that scrub through the video timeline so that the level's bubble animation is mapped to the side to side motion. The designer creates a mapping by clicking on '*Mouse*' and selecting the *move* event. The designer next defines the range of pixels that for the mouse move event destination. Clicking the "*Map to Screen Selection*" button, the designer can create a rectangular selection determining the destination mouse coordinate range. He now creates a rectangular selection overlaying the AfterEffects Timeline and ticks the checkbox so that the mouse performs a mouse *down* (holding) whenever the *move* event takes place. Because of the immediate preview, moving the phone already causes the mouse to move (which can be activated or deactivated from anywhere in the operating system using the *escape* key). As the

prototype is already interactive via its live preview, the designer immediately sees these effects (both input and output) on the mobile device (2 minutes).

9.4.6 STEP 5: FINE-TUNING THROUGH EASING

FUNCTIONS

While the interaction is being tested, the designer might find that it does not respond as desired, as it is very easy for the level to jump quickly from one side to another. One way to iterate on this behaviour is to make the bubble remain in the middle of the level for longer periods of time through an easing function. This can be achieved through an inverse cubic-in-out easing – which would slow down the animation towards the middle of the timeline, and speed up the animation towards the edges of the timeline. By playing around with different easing functions, the designer can fine-tune and instantly test the qualities of the interaction-driven animation and improve the interaction. This can take as long as the designer wishes to engage in the process. The designer may also decide to readjust the input parameters or the mouse region for further fine-tuning (2 minutes).

The prototype is now complete. The designer can modify it further as desired (e.g. its looks, additional functions, etc), or try other variations. The designer can also have others try it out.



Figure 9.11. Video based prototypes in addition to the level phone app used in the scenario. (A) shows a compass, while (B) shows a re-implementation of Android’s quick settings menu, where one can change the phone brightness.

9.5 INTERACTIVE PROTOTYPES MADE WITH ASTRAL

While the previous scenario shows one possible application of Astral, I also examined how Astral can support different types of design activities, derived from insights in Chapter 2. I specifically explore converting video into interaction-driven animations (§9.5.1), converting existing input/output into new device-specific interactions (§9.5.2), and bringing sketches to life (§9.5.3). Note that these exploration categories are meant to provide additional structure and do not imply mutually exclusive solutions to problems that designers might take when using Astral.

9.5.1 CONVERTING VIDEO INTO INTERACTION-DRIVEN ANIMATION

Video-Based Prototyping

Both our own experiences and past literature have shown designers’ inclination towards working with high-resolution video⁹ to convey prototype ideas to developers e.g., Maudet et al. (2017), Subtraction.com (2015) and UxTools.co (Palmer, 2018). While video can show state-based animations, it does not enable direct interaction. One can use a video editor in conjunction with Astral to bring interactivity to these prototypes on the target devices, in these cases, a mobile phone. I use the same workflow as the Scenario in §9.4.

⁹ Note that when I say video-based prototyping I refer to the description by Maudet et al. (2017) where designers use video editors to animate high fidelity illustrations and specify how systems should behave in terms of interface interactions and animations. This is not to be confused with the practice of video prototyping (MacKay, 1988): play-acting a Wizard-of-Oz style of prototype in video format to describe an idea.

Level Mobile Phone App. The level prototype described in the previous section belongs to this category of prototyping. In particular it emphasizes: (1) how the Sensor Selector can help designers determine which sensor corresponds to an action (in this case determining tilt by acceleration); and (2) the power of easing functions to change the ‘feel’ of an interactive behaviour.

Compass. I created a simple animation of a compass needle rotating 360 degrees (**Figure 9.11-A**), including a separately-animated needle shadow that creates a three-dimensional effect when in motion. I then created a rule that provided a linear mapping of device’s compass angle to the position on the video timeline.

Quick Settings. The Android Quick Settings menu contains a nuanced animation where multiple icons change size, position, and opacity, to reveal available operating system functions to a mobile user. With Astral, it is possible to map a downward sliding gesture to progressively reveal controls. Furthermore, one can add an additional interaction of controlling the screen brightness by mapping a side swipe on the top of the mobile screen which shows a slider (**Figure 9.11-B**) to another portion of the video timeline on the desktop in which the screen fades to black. This shows how even within video timelines Astral can support multiple interactions. While it is not a full approximation to state transitions, it is still possible to create interaction-driven animations within those limited states.

9.5.2 CONVERTING EXISTING DESKTOP INPUTS/OUTPUTS INTO NEW DEVICE-SPECIFIC INTERACTIONS

Authoring Open-Ended Interaction Techniques

With Astral, it is also possible to prototype interaction techniques that provide more open-ended ways of interaction than the video-based prototypes.

Tilt to Move. I used Astral to create a one-handed map navigation by mapping the different tilt directions from a phone's accelerometer data to the cardinal arrow keys in Google Maps. The rules are set so that key commands are triggered when the acceleration crosses a certain range (x: 4 to 7 triggers right, x: -4 to -7 triggers left, y: 4 to 7 triggers down, y: -4 to -7 triggers up). Because Astral is using a key-press event, the mapping initiates a key down when the accelerometer enters the specified range, and a key up when leaving the range. This scenario replicates an example from d.tools (Hartmann et al., 2006) that originally required programming to realize the tilt-based map navigation. In contrast, the Astral version leverages input remapping and avoids the need to write code.

Tilt-to-Zoom. I implemented tilt-to-zoom (Hinckley & Song, 2011), where a designer can pan through a map using touch, and zoom in and out via tilting provided that there is also a touch down event (their finger acts as a clutch). This is achieved using conditional constructs (shown in **Figure 9.12**). A *touch down* conditional becomes active if touch is down on the device. It contains another nested condition that checks whether *touch move* is *not* taking place. The rule



Figure 9.12. Tilt-to-Zoom Prototype. This figure shows how Astral leverages conditionals to create an interaction that requires considerable coding only repurposing a web browser on the desktop.

within this nested conditional maps the accelerometer's *y-dimension* to mouse scrolling (up or down). This prototype replicates prior research, incorporating the concept of *motion in touch* – mapping more than one sensor to a single function.

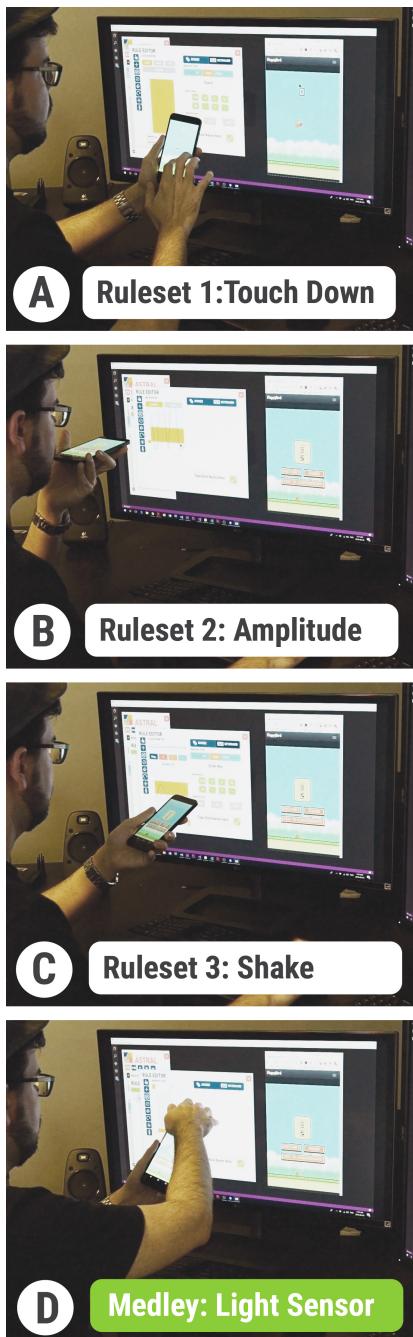


Figure 9.13. Input variations in Flappy Bird on a mobile device, using different inputs: (A) touching the screen, (B) blowing on the microphone, and (C) shaking the device. The designer alternates these by (D) covering the screen’s light sensor.

Prototyping Multiple Alternatives

In Chapter 2, I discuss how designers often generate multiple ideas and prototypes as part of the exploration process. I also described how typically many of the ideas end up stuck as an early paper sketch due to the amount of time required to realize the idea. However, with Astral, one can explore a lot of ideas quickly in high resolution.

Input Variations in a Mobile Game. Previously in §9.1, I used an example of Flappy Bird game. While working with this prototype, I mapped different mobile sensors to a spacebar keypress (shown in **Figure 9.13**) so that the bird flaps its wings when (a) tapping, when (b) blowing onto the microphone, and when (c) shaking the phone. Each of these interactions was encapsulated in individual rulesets. By creating a medley rule (**Figure 9.13-d**), one can quickly switch between active rulesets to explore different forms of interaction – in this case whenever the light sensor is covered.

9.5.3 BRINGING SKETCHES TO LIFE

Iterative Prototyping at Multiple Resolutions

Since Astral remaps inputs from mobile sensors to any key, which means it is possible to work with multiple applications at different stages of the design process. As a result, Astral can support different tasks and specialized tools – wireframing and walkthroughs, transitions between states/flow (similar to d.tools by Hartmann et al. (2006)), or working with more sophisticated programming platforms



Figure 9.14. Prototypes in low resolutions. The figure shows (A) a sketch rendered on the mobile watch, (B) a PowerPoint mockup, and (C) an in-progress drawing in Illustrator which controls the music in iTunes.

that may not be available for mobile prototyping. To realize these examples, Astral mainly relies on the *sequences ruleset type*.

Music Controller Sketches. Using a default image viewer, one can scan or photograph an interface sketch on the desktop and immediately view it (and test it) on the mobile device (**Figure 9.14-A**). Designers can emulate states by chaining multiple rules with the sequence construct. Each rule moves the streamed region to different parts of the image (i.e. the screen drawings) depending on the tap interactions that may take place. By previewing the sketches on the target device – here, a watch – designers can make early decisions such as defining appropriate button sizes.

Music Controller PowerPoint Mock-up. Presentation software such as PowerPoint and Keynote remain relevant for mocking up interfaces and wireframes (See §4.3). With Astral, it is possible to use mock-ups created with these applications to press the wireframe buttons on the watch (given the streamed visual) and move to another part of the slideshow by performing a click event on different parts of the slide thumbnail preview (**Figure 9.14-B**). Thus, one can easily test the flow between different interface screens.

Image to Media Keys. The keyboard media keys can control applications, including volume controls or music software's state (play, pause, next/previous song). It is possible to take an in-progress sketch in, say, Adobe Illustrator, and map portions of the screen to trigger different media keys. Thus, designers can switch between different songs in a music software (in this case iTunes) to simulate a remote-control software. This prototype is shown in **Figure 9.14-C**.

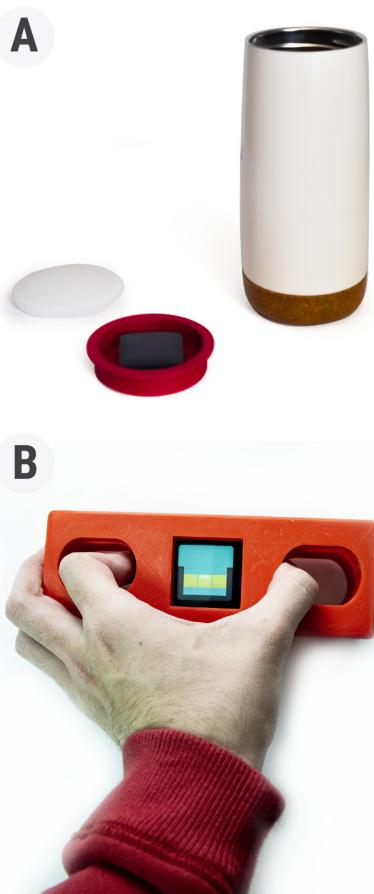


Figure 9.15. Smart object prototypes featuring (A) a mug holding a smart watch as a smart speaker, and (B) a physical prototype to the initial video-based level presented in the first usage scenario.

Authoring Smart Object Behaviours

Designers may also leverage Astral to explore behaviours on smart objects and appliances, as well as some degree of internet of things applications. Using Soul-Body prototyping (Chapter 6), designers can repurpose phones and watches in novel and unexpected ways.

Smart Speaker Animations. Using video editing applications, one can author nuanced animated responses that a smart home speaker might perform. I created a smart speaker prototype by placing a smart watch inside a mug with a 3D printed tray and light diffuser (**Figure 9.15-A**). Given this new physical prototype, it becomes possible to test different animations in AfterEffects and see what they might be like on a smart speaker. Moreover, Astral supports speech recognition through the built-in Microsoft speech API, so one could also explore different kinds of animations that trigger depending on a variety of voice commands.

3D Printed ‘Smart’ Level. I recreated the 3D-printed level from Pin-eal (Chapter 8) and reused the level mobile phone app prototype from the usage scenario described in this chapter. **Figure 9.15-B** shows a smartwatch enclosed in a larger 3D print. This demonstrates how Astral can also adapt prototypes to different devices and form factors.

With this example I also show how the prototyping tools described in this thesis can work together and assist designers in the task by creating a suite of programs.

Thus far, I showed prototypes that take different design activities at different resolutions and (1) make them interactive, and (2) bring them into the context of working with the target device. The applications in this section represent both novel and replicated prototypes (e.g. from prior research) using Astral. These examples help convey Astral's threshold, ceiling, and expressiveness (methods described in Chapter 5). **Figure 9.16** summarizes the prototypes in this section and how they are extended by Astral.

9.6 IMPLEMENTATION

Astral is designed to work with one mobile device per Astral desktop client, which constrains and simplifies the workflow. This is tied to a technical limitation of desktops, as mouse and keyboard commands only can be sent to a single focused program. The desktop client of Astral is implemented using C# and WPF, while the mobile applications are written in C# Xamarin to allow cross-platform mobile development (iOS, Android, Android Wear). To reuse code and quickly adapt to newly added sensors of future devices, all communication aspects were developed in shared code, using the .NET Standard 2.0 (see below for details).

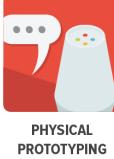
ACTIVITY	ORIGINAL DESIGN ACTIVITY	HOW IS IT NORMALLY REALIZED?	WHAT DOES ASTRAL BRIDGE?	
TURNING VIDEO INTO INTERACTION-DRIVEN ANIMATIONS	 VIDEO-BASED PROTOTYPING	<p>Video-Based Prototyping using a Video Editor</p> <p>Programming the interactive prototype (siloe platform)</p>	<p>The designer creates illustrations and animates them in a video editor on a desktop. The designer communicates with developer so that the developer can create it. Designer might comment on the end-result once it has been implemented, but iteration depends on the developer.</p>	<p>The designer does not need to wait for the developer to implement the behaviour to know how the animation turns out. The designer can try out the animation as driven by the interaction and create necessary adjustments, then show the developer.</p>
CONVERTING EXISTING INPUT/OUTPUT INTO NEW DEVICE-SPECIFIC INTERACTIONS	 INTERACTION TECHNIQUES	<p>Storyboarding and wireframing on paper or on desktop tool</p> <p>Programming the interactive prototype (siloe platform)</p>	<p>The designer creates sketches and storyboards on what the interaction technique might be like (e.g. panning and zooming on a map). Implementation is dependent on developer.</p>	<p>The designer takes an existing version of the technique on the desktop, and converts the inputs to realize new versions of the implementation on the mobile device.</p>
TESTING MULTIPLE ALTERNATIVES	 TESTING MULTIPLE ALTERNATIVES			
BRINGING SKETCHES TO LIFE	 ITERATIVE PROTOTYPING	<p>Storyboarding and wireframing on paper or on desktop tool</p> <p>Programming the interactive prototype (siloe platform)</p>	<p>The designer creates storyboards on paper showing different screen transitions. The designer might then shift to a desktop prototyping tool to create a higher resolution prototype (e.g. PowerPoint or Adobe XD). The developer will then implement it. There is no way to know if the sketches will be a good fit for the target device (e.g. whether the buttons are appropriately sized) until a fully programmed prototype shows the interaction on the target device</p>	<p>The designer can take existing results from prototypes at different resolutions and see them on the target device (e.g. a paper sketch, a PowerPoint click-through prototype).</p> <p>Designers can build physical prototypes using Soul-Body Prototyping to mock up and test a physical prototype.</p>
	 PHYSICAL PROTOTYPING			

Figure 9.16. Summary of the interactive prototypes, what the original design activity entails and how Astral bridges taking non-interactive prototypes into prototypes that can be tested on the target device and refined before implementation efforts.

Device Modules. The underlying software features classes in shared code for each of the mobile device's sensors or outputs (e.g. accelerometer, microphone, or display), referred to as *device modules*. The mobile device instantiates all modules it is equipped with when the

application starts. Once the device connects to the Astral desktop application, it sends a list of all available modules to the desktop. The desktop then creates the same modules to access the sensors by proxy, as if they were local sensors. Each module updates its values with newly measured sensor data. Modules trigger an event in code once values have been updated.

Data Exchange between Devices. Because the desktop and the client are not running on the same machine, device modules handle the internal network communication. There are two forms of communication between the clients: sensor data goes to the desktop, and desktop client visuals go to the mobile device. For sensor data coming from the mobile device, this works as follows: (1) the mobile device records the respective sensor data natively (i.e. iOS or Android specific); (2) it then updates the module using a device-independent abstraction of the measured data (e.g. three floating-point numbers for the accelerometer); (3) the module sends this data as bytes (using a unique identifier) over the network; (4) the module on the desktop unpacks the message and triggers an event; (5) if the Astral desktop client subscribes to the event, it receives the sensor data, and sends the update to rules using that sensor. Mirroring desktop contents works similarly, except that the desktop client updates the display device module. To speed up the transmission of images, we detect changes through image differencing and only transmit the areas that did change. These image (parts) are compressed (JPEG).

Performance. Astral uses wireless LAN via TCP for connectivity between devices. Astral's mobile client was tested on multiple phones (Nexus 5 and 5X, iPhones 6, 7 and 8, Pixel 2) and one smartwatch

(Sony Smartwatch 3). Image transmission is at varies depending on the image size, but typically framerates are 50 fps on iOS, 25 fps on Android. This is concurrent with mobile sensor data streaming to the desktop, yet only if the desktop actually requires a specific sensor (i.e., a *Rule* or the *Sensor Viewer* is using that sensor). Sensor data is streamed in real-time but restricted to a rate of 100 fps to ensure high transmission rates in both directions. During testing and creation of the prototypes, my co-authors and I did not experience significant delays transferring data from multiple sensors. That said, sometimes when the screen area was considerably large or on a large resolution display (e.g. a 4K display monitor), the image transfer would sometimes turn choppy from different parts slowing down. Larger images lead to the desktop screen capture and JPEG compression taking slightly longer. Similarly, the network requires transferring a larger image, and the mobile device has to then downscale the image to fit the screen size. The next section examines some of the implementation limitations in more detail.

9.7 DISCUSSION

While Astral represents a promising concept for a prototyping tool, there needs to be a critical assessment as to the extent in which the contributions have been reached (§9.7.1), describing the evaluation approach (§9.7.2), reflecting on Astral’s scale beyond prototyping (§9.7.3) and discussing some of the reality about its implementation-level constraints (§9.7.4). I suggest potential avenues for future directions, either by showing what Astral’s work paves the way towards, or by suggesting potential improvements to the system or design practices in general.

9.7.1 REVISITING THE DESIGN RATIONALE

Earlier in §9.2, I described the design rationale behind creating Astral presented as four decisions which shaped its contribution. It is important to have a closer look at the extent to which each one of these objective have been achieved and discuss potential extensions.

R1. Prototyping Live Interactive Behaviour on the Mobile Device.

Astral, as shown by the scenario and prototypes, indeed supports live authoring, and it is possible to test the behaviours as they are being created. In fact, this proved to be very useful in many research discussions with colleagues and co-authors, as it was possible to adjust the rules to reach the desired effects almost instantly. However, there are a few caveats to consider, namely: it is possible to create conflicting rules, mappings can sometimes accidentally affect the wrong application, and the inputs can become locked (described in §9.3).

The current implementation of Astral does not check for conflicting rules. This has been a deliberate choice to ensure designers have more flexibility, but also opens the door for potential sources of human error. For example, a designer can create two rules that are nearly identical but map the mouse movement in opposite directions. The result will be that whenever the rules are both active, the mouse cursor will start rapidly jumping back and forth. That said, given that the interactive behaviour explorations (such as the ones in the prototypes above) take only a few rules (none of the examples have more than five rules running at once), designers should in theory be able to keep track of what they are creating. Rather than adding constraints to the interface (e.g. showing a warning in a suspected case of conflict), I consider a better solution is to further iterate on the way the

list of rules is displayed to increase the amount of awareness cues provided, so that designers are more aware of the effect of their actions.

The next consequence of live authoring with mouse and keyboard commands is the possibility for inputs to affect the wrong application. Given that Astral is unaware of the active application and simply executes mouse and keyboard commands on the fly, there is a chance that a designer might perform an accidental undesired action. For example, a mouse click could accidentally activate another application which will receive the remapped inputs. This in fact happened to me a few times before I implemented the '*escape*' key workaround, where I would have a small window of a web browser on top of a full screen version of Astral's code. I would then create a behaviour that performed a click and drag operation. In the behaviour creation process I would unwittingly move the web browser window to read the underlying code without deactivating the rule. I would then lower the phone, which makes the accelerometer move and triggered a click, which would click and drag on top of my code window, thereby selecting my code and rearranging it to a different location. Perhaps a stricter implementation could enable selecting whether the mouse and keyboard events should affect every desktop application, or only affect a subset of the currently opened applications, but it would add extra steps to setting up Astral.

The last and perhaps least concerning (yet most common) consequence of live authoring were the input locks. Input locks occurred

sometimes in the process of creating a rule which mapped a very sensitive sensor (e.g. accelerometer) to the mouse cursor movement. The system would start a live preview immediately once the mouse cursor mapping was established, which could in some cases prevent one from regaining control of the mouse again. The ‘*escape*’ key solved this situation entirely right away, as the moment an input lock took place it could be temporarily turned off with one key. Because the system listens for the ‘*escape*’ key across the operating system, it also became extra useful for live demos. With the ‘*escape*’ key it became possible to author the rules, minimize everything, and then play the rules while showing only the target application.

As long as designers are aware of these potential side effects of live authoring, and that they know that the ‘*escape*’ key can relax some of these effects, I expect there to be few problems beyond usability details. I think that because Astral has so few added menus and constraints it works well as a tool that one can use quickly without having to do too much setting up (e.g. defining which application to affect). Given the current design, I do believe that Astral could become a quick “*walk up and use*” tool once learned. That said, I think that it would be beneficial to provide additional feedback so that designers are more aware of the current mappings and the effects of their actions on the mobile device. The active rules in the main view of Astral could show live previews along with the source input from the mobile device and the destination input on the desktop, as well as which area it will affect. One way of doing this is envisioned in **Figure 9.17**.

R2. Providing an End-User Interface that Allows Designers to Explore Variations Among Mobile Sensors. For Astral to be a tool that

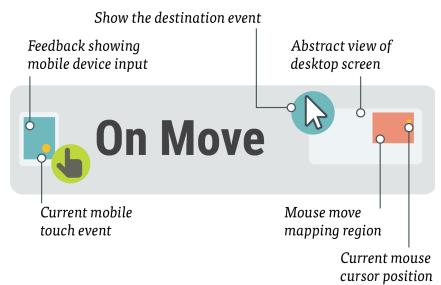


Figure 9.17. Rule preview enhancements with additional feedback and awareness cues.

designers can use quickly and effectively, there needs to be as little coding as possible (for the discussion of this problem see Chapter 4, §4.4.2). Thus, the visual representations for sensor data need to make sense as much as possible. I put care into trying to achieve a strong *expressive match* (Olsen, 2007) by tailoring visualizations to particular mobile sensor data through the Sensor Selector interface. However, in spite of a higher *expressive match*, there is no guarantee that there is a reduced need to learn and understand how sensors work. For instance, consider that accelerometer data can be used to determine the orientation of a mobile device. While visual inspection on the Sensor Selector could lead a designer to realize the effect of device tilt on the accelerometer values, it is hard to know without understanding how the sensors work, whether it is a generalized effect. While one could study how designers learn to work with sensor data and the extent to which tools need to explain it, I believe this is more of a communication problem. More of the existing programming-specific documentation needs to reach designers and even computer science students.

R3. Supporting the Use and Repurposing of Existing, Familiar Applications for Prototyping. The prototypes discussed in sections §9.4 and §9.5 show a variety of applications which match many of the tools used by designers in the past ten years (see §4.2). These range from more general usage tools such as the web browser, to more casual ones such as PowerPoint, to more specialized tools such as Illustrator and AfterEffects. What is interesting is that Astral, in cases such as the video editor example (§9.4), can take a non-interactive video on the desktop and instantly turn it into an interactive experience on the mobile device. The fact that the area that is mirrored to the mobile

device screen (AfterEffects' video preview) is different from the area in which the mouse move event is being executed on (the timeline) suggests that there might be a lot of room for interesting applications of Astral beyond what is presented in this chapter. Thus, even though it seems like the *ceiling* of Astral only goes as high up as the desktop application itself, the reality is that Astral, unpredictably, achieves more than that, as it now accomplishes functions that were not possible in the native application.

R4. Supporting Interaction-Driven Animations. An important aspect often overlooked in interaction design prototyping tools is the authoring of behaviours which respond to continuous input as the interaction takes place. Astral supports *interaction-driven animations*, provided that the desktop application can create some form of moving visual, such as the timeline preview in AfterEffects or the scrolling to zoom in on the web browser maps. However, with such tools it is only possible to progress through one dimension of output: the video player has a single timeline, the zoom function on the map is dependent only on the mouse scroll. The current types of animations likely cover a majority of the common interactive behaviours. Still, it would be interesting to animate individual objects as a function of different sensors, or to manipulate different timelines at once. This could lead to richer behaviour explorations perhaps more in tune with other areas such as game design (e.g. using two joysticks to affect the animation and response).

9.7.2 EVALUATION APPROACH

There are various strategies for evaluating toolkits as discussed in Chapter 5. Of these, I use *evaluation by demonstration* as the primary

method. In particular, the prototypes tie back to key design activities as reflected in Chapter 4, where Astral can bring the prototype to the target device in a way that it can be tried out, thus enabling further elaboration of interactive behaviour. The prototypes represent both novel and replicated systems from past research, which reflect how Astral can achieve results that might be difficult to create otherwise, as well as ensuring that prior *paths of least resistance* (Myers et al., 2000) can still be accommodated. The usage scenario (§9.4) provides a perspective on how designers might work with Astral while conveying some of its *threshold* and *ceiling* (Myers et al., 2000). More importantly, I have carefully considered the claims made in this chapter and revisited the original goals.

It is important to note that the potential research question for a hypothetical future study is not whether designers can use Astral. The problem with such question is that it easily focuses on usability bugs that get in the way of using the system, and there is little inquiry on the value of Astral as a potential design tool. The larger question and more interesting question for future work is more about *how designers might leverage a tool like Astral*. This question is a much more difficult to answer, and studying it directly with designers is beyond the scope of this thesis. This question, however can still be answered through the demonstrations in §9.4 and §9.5.

I deliberately did not pursue a lab usability study. A usability evaluation would be inappropriate given that Astral is not a walk-up and use system and the paths of interaction are very open-ended (Olsen, 2007). Furthermore, a lab study would sacrifice realism (McGrath, 1996). There are three reasons why it is so: First, designers each have

different applications and computer setups which cannot be reflected in a lab setting. Second, Astral provides an alternative way to think about prototyping, which requires time to internalize. Finally, short tasks can lead to the “*usability trap*” (Olsen, 2007), or test tasks we know Astral can succeed at, thus leading to unfair comparisons or weak generalizations from the current implementation rather than the concept as a whole (Greenberg and Buxton, 2008). Open-ended tasks require designers to envision ideas ahead of time (thus requiring an understanding of what Astral can do) – it would be unreasonable to request a design on the spot.

One interesting avenue for future work is to consider an observational field study, as it might be an appropriate way of testing Astral’s potential for designers. However, for the software to be deployable would require a large amount of testing and additional engineering efforts, which are beyond the scope of this thesis.

I see two large issues in a potential user study with Astral and both stem from the existing design practices. The first challenge is that designers have a specific set of expectations which are a consequence of the tools of today. The current set of tools and job descriptions typically guide a designer towards creating wireframes and specifications without a lot of room for exploration and animations. Moreover, because of the lack of tools today, little is known about what a smart object behaviour designer might look like. Thus, the number of potential people that could truly experiment with Astral for mobile interaction or smart object design outside of the research community is perhaps small if not still nonexistent. This becomes evident when looking at the plethora of mobile applications and how aside from mobile

games and operating system functions, the explorations of nuanced behaviours are quite limited. The second challenge is that a tool will always be compared with the current industry software, and measured with the expectation that it achieves similar levels of complexity. This is difficult to achieve with small prototypes like Astral. I think the solution is for Astral, or a tool like Astral, to become robust enough and readily available so that people can try it, or bring it into an educational setting with enough support and documentation. For example, perhaps motion design students could see how their ideas can be applied in the context of interaction design.

While Astral could enrich interaction design practice, Astral is not intended to replace existing prototyping software, but to instead provide an alternative approach. State models, as discussed in Chapter 4 §4.3.3, are a common prototyping strategy, and there is a reason they remain standard. State diagrams can quite intuitively describe the flow of the interaction. However, explorations with state models can easily be complemented by leveraging Astral's *closed loop of interaction* to explore how behaviours might play out on different target devices, or to try out new interesting *interaction-driven animations*.

9.7.3 SCALE AND REAPPROPRIATION OF TOOLS

When looking at Astral's application agnosticism, I realized one reality in current technology is that development platforms are largely siloed and highly specialized. For example, to develop for smartwatches, developers can only work with Java on Android Studio, Swift in XCode, or Xamarin in Visual Studio. With an application like Astral, one can potentially bridge development for short-term use applications. Moreover, from a conceptual standpoint, Astral is a plug-

and-play application, which means that there is no need for custom software IDE and API installations. A designer can work with PowerPoint or go as far as to create an HTML/Javascript site that runs on the watch.

While this chapter focuses on Astral as an interaction design tool, it is worth mentioning that its applications can go beyond what has been shown. For example, an early prototype I created served to control applications remotely. Applications such as Adobe Illustrator do not have hotkeys for all features, such as alignment tools (Figure 9.18). Consequently, designers often make selections and drag the mouse cursor to an edge of the screen to select a command such as aligning all objects to the left. When conducting repeated alignment tasks, the constant moving back and forth of the mouse can become tedious. One way to solve this is to move the interface to a mobile device using Astral. This way, the interface can rest on the phone by the desk next to the keyboard and provide end-users with opportunities for bimanual interaction: the mouse on the right hand selects the objects on the screen while the left hand chooses the desired command. Moreover, unlike keyboard hotkeys, the distributed interface shows meaningful icons that inform the user of the effect of the action. This experiment shows a replication of prior visions of using mobile devices for control (Myers, 2002) and creating custom interfaces by borrowing from existing ones (e.g. Stuerzlinger et al. (2006)). Thus, a future avenue is to explore how Astral could be adapted for people to create custom interfaces and extend their current application functionality.

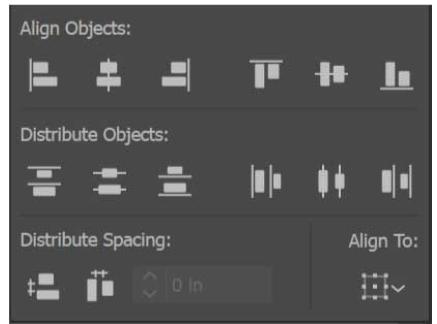


Figure 9.18. Alignment tools in Adobe Illustrator.

9.7.4 IMPLEMENTATION-LEVEL CONSTRAINTS

It is important to remember that Astral is in itself a proof-of-concept prototype, developed by a small team of researchers in a limited amount of time while handling different projects. There are certain limitations, or constraints, in terms of the current implementation. Some of these details are less relevant to the concept and more related to the usability, while others helped shape more heavily the expectations a potential user might have when using Astral. I focus on three aspects: the challenge of the complexities behind Astral, the fact that each desktop and mobile device is different, and the degree of permanence of a prototype made in Astral.

Implementation Bottlenecks

One aspect that is important to stress is that Astral itself is a highly complex proof-of-concept system with many components working together. The display capture code leverages libraries in .NET graphics in *System.Drawing* that are at least 10 years old, which are not highly prioritized within the operating system (thus capping at 60 frames per second). Once images are transferred, mobile devices across multiple platforms which treat images differently (e.g. Android is big endian while iOS is little endian, so individual bytes from images that arrive need to sometimes be reinterpreted, which might explain some of the bottlenecks on Android). Xamarin, while robust and capable of generating cross-platform shared code in C#, is not as optimized as using native languages such as Swift (iOS) or Java (Android). Sensor data across devices varies from model to model. The data is inconsistent in terms of which sensors are available, their re-

fresh rate, and the levels of granularity (as well as the range of maximum and minimum values). While most of the software is written from scratch, there are many external libraries that we leverage, such as to retrieve the camera view in WPF. This “daisy-chaining” of libraries on a prototype leads to issues such as memory leaks, which can slow down the system over time. Moreover, the rule system, visualizations, and interface are all built from scratch, and were the result of an evolving architecture which was updated as new ideas emerged. As a result, Astral also lacks key features that enable longer term usage, such as inability to save rulesets once the software closes, the absence of an undo-redo stack, or the inability to reconnect when the system crashes. Lastly, the open-endedness when creating rules means that there needs to be extensive testing. These tests might include orders of operation across different rules, as well as error checking when settings for a rule are left blank. These aspects matter more if Astral were to become a product, so they are relevant for the implementation but have no bearing on the concept itself.

Astral’s implementation limitations are a direct result of integrating many specialized features of networking, image processing, mobile sensing, data visualization, operating system hooks, etc. It is important to note that Astral works well enough to realize the prototypes described thus far, as well as different permutations, provided that the user has a good understanding of the code-base (i.e. they are resilient to bugs and crashes). This is not an uncommon challenge in systems research (Olsen, 2007) as it is unreasonable to expect a small team of researchers to create a full product. Astral in the end remains a prototype in its own, and taking it to a stage in which it can be deployed onto different machines reliably would require significant

time, engineering effort, and a larger highly proficient team. As such, making Astral deployable is beyond the scope of this dissertation.

Device Relativism

Every device is different, and Astral works with values relative to both the desktop and the mobile device being used.

Mappings of mouse and screen coordinates may not work across different computers with different resolutions, as all these are relative dimensions for each desktop monitor. Moreover, desktop application window sizes are not fixed, so once the workspace has changed the mappings may no longer work. There are workarounds to the latter concern: it is possible to store the position and sizes of the windows and associate them to the rules, so that when a ruleset executes it adjusts its values to match the window sizes. Yet, storing this information still may not carry to another desktop, with different resolutions, zoom factors and customizations within the familiar applications (e.g. where controls and palettes are placed within the window).

Mobile phones and smartwatches also have a wide variation within their resolutions and sensors. Additionally, some sensors may not be available on each device, and some sensors may have device-specific readings in units that may not be intelligible. One example of an unexpected sensor behaviour was that the Nexus 5X phone has a very sensitive gyroscope which updates at a very fast rate with very fine-grained values (over 6 decimal points). Transmitting this data on updates caused the system to slow down and eventually crash, which led us to add a threshold that can be set in code to determine how much change there needs to be before value updates are sent.

Permanence

The device relativism and current implementation lead to a side effect of Astral, which is that the prototypes are only temporarily functional. Prototypes work as long as all external software is running and the active application that the designer wants to manipulate (e.g. AfterEffects) has not been moved or resized. In other words, Astral is not aware of what applications are being manipulated, it is simply a means to provide mapping from mobile device data to the desktop while having a mirrored view of the desktop screen (or a portion of it). Otherwise, the designer needs to readjust the values in each of the rules. The lack of permanence is not necessarily an issue, as it ensures the designer focuses on the behaviour design in itself and keeps the activity as a self-contained exploration. Given that Astral can be simply opened and operated on top of the existing designer workspace, the cost of setting up the interactivity for the prototype is relatively low. That said, the problem is that it does not leave a lot of room for simply reusing the prototype or comparing different states or versions within it. One interesting avenue of future work would be to store the streamed images along with sensor values into the mobile device to create small application simulations. With this, the prototypes in Astral could be saved for longer-term usage and for more communications within the larger design and development team.

9.8 SUMMARY

This chapter presented Astral, a prototyping tool that allows designers to create rich interaction driven behaviours for mobile devices by repurposing existing applications. Designers can stream contents of a desktop and reflect them on mobile devices, and easily create rules

that map mobile input into desktop mouse and keyboard events to enable the repurposing of existing applications. I described the interface in Astral and explored a series of prototypes that demonstrate how the tool can be used in different activities within the design process. In particular, Astral integrates aspects of Soul-Body Prototyping discussed in Chapter 6 and shows how designers can create interactive behaviours for smart objects without the need to code. Thus, Astral serves as the glue to bring together my thesis that we can *repurpose existing hardware (mobile phones and watches) and software to enable designers to create live interactive prototypes for smart interactive objects.*

CHAPTER 10. CONCLUSION

This conclusion chapter revisits and reviews the solutions to the problems outlined in Chapter 1, addressing my thesis, which is:

we can repurpose existing hardware (mobile phones and watches) and software to enable designers to create live interactive prototypes for smart interactive objects.

I first revisit the designer challenges discussed in Chapters 1 and further developed in Chapter 6 (§10.1). I then discuss the primary (§10.2) and secondary (§10.3) contributions of my work before sharing some potential future directions (§10.4). I reflect on the overall role of this thesis (§10.5) which shows my current overall vision and where this work might continue.

10.1 REVISITING THIS THESIS' TARGET PROBLEMS

In Chapters 1 and 6, I describe three core problems that designers face which come about understanding the background and training of

interaction designers, as well as the existing tools to author interactive behaviour (described in Part 1 of this thesis). Specifically, when prototyping smart interactive objects, the challenges I outlined are:

- Need for multiple specializations, specifically programming, circuit building, and form-giving
- Lack of tool support beyond click-based screen transitions
- Need for close-to-product representations, where designers can manipulate the physical objects and feel the interactive behaviours in context

The Soul–Body Prototyping paradigm, along with Pineal and Astral as tools that operationalize it, directly address these three designer challenges. In Chapter 6, where I propose the Soul–Body Prototyping paradigm, I explain how mobile devices can replace the need for electronic circuits through their many sensors and outputs which can perform many of the functions of smart objects. Moreover, these constructs are brought to higher levels of programming, as the sensors and outputs have additional software abstractions that facilitate the development such as event-driven programming, as well as platforms that can automate many of the time-consuming low-level programming (e.g. memory management). These development benefits make it so it becomes easier to develop even higher-level design tools to exploit these different sensors and outputs, as demonstrated by Pineal (Chapter 5) and Astral (Chapter 6). Given that mobile programming platforms are well maintained, it also means it is possible to tighten the design-build-test cycle, such as through live programming, where designers can see the effects of their decisions right away

as opposed to having to compile code, reassemble circuits, etc. to test the behaviours. In addition, since Astral leverages existing tools, it means it is possible to go beyond simple screen transitions, and tackle aspects of interaction-driven animations. While Soul–Body Prototyping can be carried out through many kinds of materials (e.g. cardboard), Pineal’s fabrication-ready 3D models mean designers can achieve a closer-to-product representation, which in conjunction to Astral’s high ceilings for software sophistication, can lead to prototypes that designers can manipulate, feel, and fine-tune. This is not to say that these are the only two solutions, but instead to show that with two self-contained explorations it was possible to see a relatively expressive outcome through a variety of prototypes. Indeed, there are more opportunities for better understanding interactive behaviours, and different authoring mechanisms and paths of least resistance that may favour different end-results and classes of prototypes.

Both Astral and Pineal are early explorations, which raise the question of how designers might actually work with these kinds of tools. This is a difficult question beyond the scope of this thesis, and one that may not necessarily have a right answer. Many of the design tools in HCI research promise hypothetical futures, but the only way to truly know if these tools will indeed change people’s practices is by becoming part of the practice as fully fledged tools with ongoing support. Laboratory experiments can only answer some questions specific to the implementation details of these systems, such as usability problems at the interface level. This truly leads to a paradox in our field of research, as technologies that become common-place can shape existing practices, but for practitioners to adopt new technologies, these technologies need to become common-place. In that

sense, the role of an interaction designer is exclusively defined by what they can create as a common denominator with the tools that are available. Perhaps an interaction designer for smart interactive objects does not exist as a job description because there are no tools that let an interaction designer create these kinds of prototypes. Today's design tools (discussed in Chapter 4) seem to converge towards large standardization of interfaces and little opportunities to diverge and customize outside of these boundaries.

This dichotomy of practice and vision are a common challenge in HCI as a field. My approach has thus been to try and best understand both the audience of these tools, in this case interaction designers, as well as what the tools both in research and industry themselves offer. As a result, it becomes possible to create certain constraints given the understanding, such as acknowledging that there need to be programming alternatives that do not require coding. From there it becomes possible to envision new tools. While these tools may not become part of everyday practice in the short term, these ideas can then live and inform future systems both in research and industry. This is why, as discussed in Chapter 3, demonstrations are crucial components to exploring what software tools might do, and realizing bold visions of the future, as revealed by the many toolkits we surveyed. Similarly discussed in Chapter 3, it is key to determine whether a user study would add significant value to the work beyond a “sanity check” and realize that as researchers we are in a much stronger position to critically envision potential future practice as opposed to our study participants.

THESIS STATEMENT EXCERPT	RESEARCH QUESTION	GOAL	SOLUTION	EVALUATION
We can repurpose existing hardware (i.e. mobile phones and watches) and software (i.e. desktop operating system functions and applications) to enable designers to create live interactive prototypes for smart interactive objects without the need for code or custom circuitry?	How might we devise a means for designers to author interactive behaviours for smart interactive objects without the need for coding or custom circuitry?	To explore and devise a means for designers to author interactive behaviours for smart interactive objects without the need for coding or custom circuitry.	A prototyping paradigm which proposes repurposing mobile devices and existing desktop software to create smart object prototypes. This paradigm is then explored and operationalized through two prototyping tools.	See below.
1. I explore how designers can use mobile devices and their built-in sensors and outputs in place of electronics to author interactive smart object prototypes	RQ1. How might designers repurpose mobile devices to prototype smart interactive objects?	I will devise a means for designers to repurpose mobile devices to prototype smart interactive objects.	The Soul-Body Prototyping Paradigm. This paradigm suggests leveraging the functions of mobile phones or watches as the “soul” of an object, and building a “body” around the device to give its meaning. Consequently, designers can leverage a mobile device’s inputs, outputs, and internet connectivity to create new smart object prototypes without the need to create custom circuitry.	<i>Usage.</i> Early computer science undergraduate student explorations which demonstrate the opportunities and relative ease of development using mobile devices in conjunction with Phidgets electronics.
2a. I create proof-of-concept prototyping tools that build on top of existing desktop applications to create physical prototypes that enclose the mobile device in an appropriate form given interactive behaviour specifications.	RQ2. How might designers author forms around mobile devices to make them look and feel like smart objects?	I will repurpose an existing 3D modeling tool to aid in the creation of forms that can fit mobile devices.	Pinhead. A prototyping tool in which trigger-action behaviours automatically modify a 3D model. The programmed behaviours act as instructions to create a fabrication-ready form which can fit a mobile device and expose the relevant inputs and outputs.	<i>Case Studies.</i> Two prototyping tools (see below) and their resulting prototypes which operationalize the paradigm and remove the need for electronics and coding when creating smart object prototypes.
2b. I create proof-of-concept prototyping tools that build on top of existing desktop applications to repurpose familiar desktop tools to author rich interactive behaviours that are driven by the user’s interactions.	RQ3. How might designers leverage existing familiar software tools to author interactive behaviours for smart interactive objects?	I will devise a means for designers to author interactive behaviours for mobile devices using familiar desktop tools.	Astral. A prototyping tool in which designers can mirror a portion of the desktop screen onto a mobile device. Designers can create simple rules that convert live mobile device sensor data into mouse or keyboard events. As a result, the mobile device can remote control familiar desktop applications and operate the application in the context of dynamic interactive behaviour prototyping.	<i>Demonstrations.</i> A series of novel and replicated physical prototypes which represent the design space of Soul-Body Prototyping. A usage scenario helps showcase the process of working with the tool as well as its threshold and ceiling.
				<i>Demonstrations.</i> I show a series of interaction design prototyping activities as carried out by Astral using a variety of applications. These demonstrations include both novel and replicated examples, including some that are not currently possible to achieve without programming (e.g. tilt-to-zoom). I also show how a designer might transition from a video to an interactive prototype through a usage scenario.
				Additional performance benchmarks for framerate, and reflective discussions which include aspects of Myers et al. (2000) and Olsen (2007).

Table 10.1. Summary of primary thesis contributions

10.2 PRIMARY CONTRIBUTIONS

This thesis at its core devises an alternative way for designers to prototype interactive behaviours for smart interactive objects without the need for coding or custom circuitry. In doing so, I examined both conceptual and technical standpoints. I summarize the three primary contributions in context with my thesis statement, research questions, goals, as well as evaluation in **Table 10.1**.

Conceptual Contribution: Soul–Body Prototyping. The Soul–Body Prototyping Paradigm provides an alternative way of thinking about designing smart interactive object prototypes by repurposing readily available smart phones and watches into a new context. I provided a design space of the different mobile inputs and outputs as well as ways in which they can be repurposed for physical prototyping. In particular, the categorization of modifiers (rerouting or transducing) opens up ways to think about creating new and interesting physical widgets for potential interactions with smart objects. The collection of individual prototypes across Chapters 4, 5, and 6 are all living examples and demonstrations of the opportunities of the paradigm.

Technical Contributions: Pineal and Astral as Software Tools. The two proof-of-concept systems created for this thesis, Astral and Pineal, are ways of operationalizing Soul–Body Prototyping and bringing the paradigm into the interaction design context. The sample prototypes created with these tools showcase the breadth and ceiling of what is possible to create in terms of interactive behaviours that can be tested live on a physical prototype. These prototypes are created without the need to code and without having to create custom circuitry, which on its own removes several time-consuming steps, and

enables designers to focus on the design activity itself rather than the implementation details. There are also individual features of the tools that are novel and could be considered in future design tools, such as Astral's visualizations which are tailored to specific sensors, with live updates, as well as the ability to record and scrub through video synchronized with the sensor data. Pineal and Astral are tools that could be used in combination and suggest that perhaps there could be a larger suite of tools that support different aspects of Soul-Body Prototyping that may be suitable at different stages of the design process.

10.3 SECONDARY CONTRIBUTIONS

Interaction Design Background. Chapter 2 presents a comprehensive review looking at interaction design as a field, as well as prototyping theory, with a large corpus of work that has emerged since the last comprehensive review of the field by Hartmann (2009).

Defining Interactive Behaviour. In addition, I integrate different theories from both interaction design research and HCI to start uncovering what researchers might mean when referring to interactive behaviour, and examining the different authoring approaches to these types of prototypes as done in research and industry tools to date.

Toolkit Evaluation Methods. The survey on evaluation methods for toolkit research represents an empirical overview of different approaches taken by researchers to date, which broadly includes demonstrations, usage, performance and heuristics. This perspective helps support prior criticism to the role of usability studies, as noted by authors such as Olsen (2007), Kaye (2007), as well as Greenberg

and Buxton (2008). The collection of methods provides an initial understanding as to how toolkit researchers create and validate knowledge within the field of HCI. More importantly, this work and its approach has inspired similar kinds of surveys in other areas of HCI, including the landscape of creativity research (Frich et al., 2019), as well as the taxonomy of cross-device interactions (Brudy et al., 2019). It is worth emphasizing that Brudy et al. found that these same evaluation strategies I uncovered in Chapter 3 also carry out to over 300 systems investigating aspects of cross-device interaction, further generalizing this knowledge.

10.4 FUTURE WORK

Given the set of building blocks in any software tools, they are bound to shape how people think about, and approach problems, as well as what is possible to produce. As a result, having a wider variety of methods and tools than what already exists can lead to different kinds of solutions and accommodate individual ways of thinking. Soul-Body Prototyping and the current encompassing tools already provide an alternate way to think about how to use mobile devices for prototyping, how to prototype smart objects, and how to realize interactive behaviours. The next set of projects propose a natural extension to this dissertation's work, while others suggest further questions for the field.

10.4.1 MAKING EXISTING OBJECTS INTO SMART PROTOTYPES

Soul-Body Prototyping thus far has examined making new objects from scratch. An alternative thought is to consider what if one could

attach a mobile device to an existing object to immediately turn it into a smart object. This can be a prototyping approach that leads to new kinds of smart objects, or new ways to think about smart objects. After all, mobile device sensors and outputs provide a creative prompt to generate new kinds of ideas. For example, a mobile device can be attached to a door to create a smart door: the motion patterns can know if the door is being opened or closed, while sound patterns can suggest if the door is being locked or unlocked. One can then experiment with different ways of displaying notifications on the door itself (e.g. if someone knocked the door while one is away), or perhaps sending messages to the homeowner's phone (e.g. if they forgot to lock the door). One could also attach a smart watch to a coffee machine and detect when it may need cleaning, or track how often different coffees are made. The lack of actuation and the constraint of the mobile device can also provide an advantage, as it can foster more communication between the end-user and the device to better understand what is happening at a given moment. Thus, this type of Soul-Body Prototyping opens opportunities to new kinds of smart object prototypes that can communicate with people or track activity without taking away people's agency and control. Additionally, one can explore how to create architectures that can help make some of these prototypes into longer term objects, and provide people with novel ways of reusing old mobile devices. Some examples of smart objects and how they can exploit mobile sensors and outputs are summarized in **Figure 10.1** as a sketch.

10.4.2 SOUL-BODY MULTI-DEVICE ECOLOGIES

In Chapter 1, I discussed how one can think of the interaction with a single device, or one can look at how one or more devices support people's activities. A natural extension of the body of work in this dissertation is to consider how to create prototyping tools that leverage multiple devices. One can think of how individual devices interact together, meaning that the input of one device yields input in another device, and then further expand considering how we can define scenarios with activities. Buxton (2018) discusses the concept of place

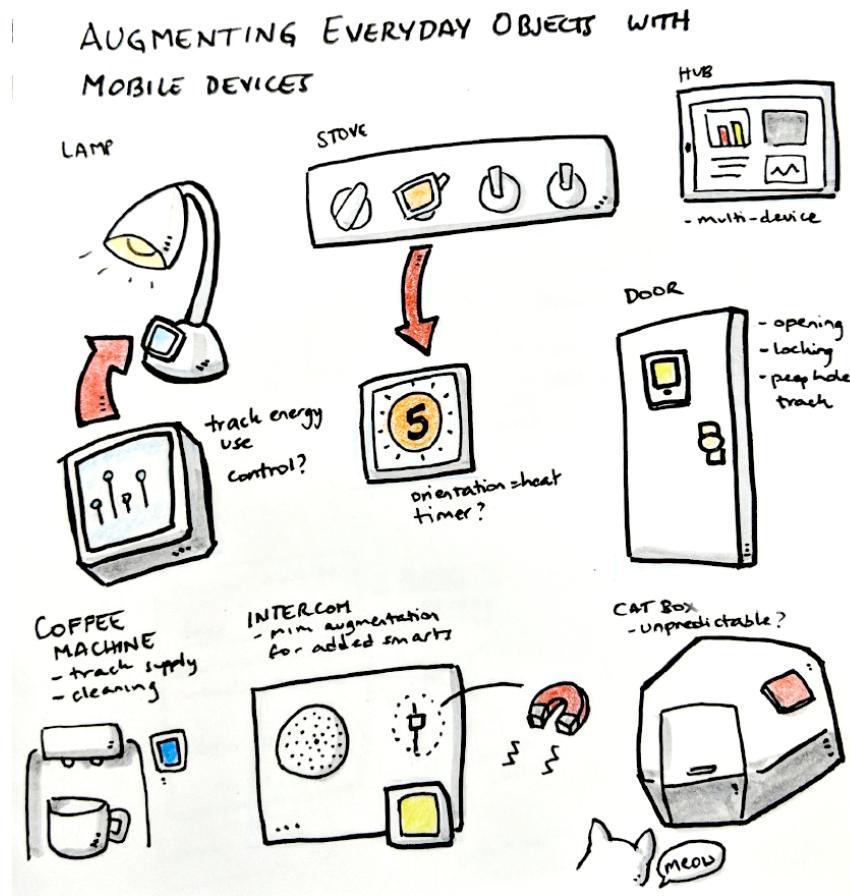


Figure 10.1. Sketch sample showing different ways of augmenting existing objects with mobile devices to work as smart object prototypes.

being more than a single location, and exemplifies his concept of *ubiquity* through a personal digital assistant, where the conversation can move as one travels from one room to another by connecting to different smart speakers in different rooms. In that case the interaction is no longer about the device, but the seamless transition from one location to another while the interaction still takes place. This exploration can initially look at how devices communicate with each other, or how multiple devices can incorporate themselves together within a single object to provide different points of sensing and outputs. With these, it is possible to create more complex prototypes. For example, consider a smart home where opening a door gradually makes the lights turn on as the door is being opened. Another example could be to build an air hockey table where mobile devices are attached to the two goals and use their sensors to track the score. As a result, the scale of applications increases and thus can lead to user experiences where the devices are working more interconnected while still supporting everyday human activity.

10.4.3 SUPPORTING MULTIPLE PARAMETERS

With current prototyping tools, and even with Astral running a video editor, only one “main” animation can take place at a certain point. Yet, this is not how many interactions play out. For example, Astral’s implementation of *tilt-to-zoom* (§9.5.2) can animate more than one parameter at once: the position (x, y) from the touch event, and the zoom from the accelerometer tilt. This is one of many examples in which the input leads to an animation that affects more than one output parameter. The question that arises is to devise *how designers can*

author more complex experiences in which multiple sensors are tied to a variety of outputs which are independent from each other. One way to do so is through a more generalized animation tool, which instead of a main timeline it features a series of independent sensor-based timelines from which one can create keyframes and motion tweens. Thus, designers could take multiple sensor data and associate individual sensor values to independent animations, generating more dynamic and responsive environments. The animations could span both motion tweens (explained in Chapter 4 and 9) or frame-by-frame animations.

10.4.4 NEW BUILDING BLOCKS FOR INTERACTIVE BEHAVIOUR DESIGN

One potential way to break from existing authoring approaches is to generate a new one by studying and analyzing how designers describe interactive behaviours. Myers et al. (2004) conducted a study in which they devised programming building blocks by asking programmers and non-programmers to verbally describe photos showing different states of a program. One instance included, for example, an image showing Pac-Man moving and stopping when hitting a wall, where participants had to then verbally describe what a program should do. This study led to rich insights, such as the common use by programmers of the word “*when*” to describe events. “*When*” has been adopted as a keyword in newer programming languages such as Scratch (Maloney et al., 2010), and can be considered a simple concept to understand: *when an event happens, do something*. A similar approach to the study by Myers et al. (2004) can be extrapolated into design, where one could study the kinds of words and descriptions

designers use given a variety of prompts. Moreover, it is also an opportunity to assess the different approaches designers might take to solve these prompts in a variety of resolutions, to better understand how they think about certain interaction problems. One could select a variety of interactions that populate a representative portion of the interactive behaviour framework in Chapter 3, and have designers describe them and then express how they would prototype them in different ways. Careful consideration would have to be placed to make sure the designers are not biased by the prompts, that is, ensuring that a particular authoring approach is not in any way implied or suggested. This can be mitigated by using visual narratives (e.g. wordless comics) or even videos. While there is a possibility that many of the results replicate what is already known, we can discover some of the designers' inclinations, as well as more fine-grained details of how they express and describe what happens.

10.4.5 REFLECTIONS IN SYSTEMS RESEARCH

The work in toolkit evaluation made me realize that a lot of the reflection on toolkits is the result of the authors' experiences, and that some of the richest validation was from their insightful discussions rather than their studies. Many of these studies have taken Myers et al. (2000), as well as Olsen (2007), and leverage their work as a vocabulary to discuss certain aspects of toolkit research such as the expressiveness or the flexibility. At this point in time given a larger corpus of work, there is room for devising a broader vocabulary and set of questions authors can make when looking at authoring environments. For example, Myers et al. (2000) bring forth the concept of "*threshold*", which refers to how easy it is to get started with a tool.

However, one can further discuss the threshold in terms of different perspectives. For example, “*technical threshold*” can refer to a user’s ability to get started with the software, while “*setup threshold*” can refer to how difficult a tool can be to put together so that the user can operate it. A “*conceptual threshold*” may refer to the required expertise of the user in terms of the concepts they are exploring with the particular tool.

These discussions typically stem from the systems’ demonstrations. While I argue that demonstrations are a valuable form of evaluation, another important question that remains is what makes a strong or weak demonstration. It is possible to distill what demonstrations do to support the overall argumentation. For example, Astral’s demonstrations felt unique in that they establish how to fit within existing practices, but also how Astral as a tool can co-exist with the applications today. In that sense, a new interaction technique may show more value by showing how it integrates into todays workflows and environment within a device than testing its accuracy and speed. Somehow this type of reflection shows that a research vision can be realized and can belong in our existing ecosystems.

10.5 REFLECTION

Together, the contributions of this thesis work entail that interaction designers can now join in and perform activities that were not possible with current tools. The existing literature, as discussed in Chapters 2 and 4, reflects that interaction design tools in the market today do not align with interaction designers’ main goal, which is to explore and envision interactive behaviours, largely due to the technical limitations of what the tools can do. Soul–Body Prototyping, together

with Pineal and Astral, as presented in this thesis, show how their applications can adapt to tasks designers already do, and even co-exist with already available hardware (phones and smartwatches) and software platforms (e.g. 3D modeling tools, video editors). The focus on interactive behaviour proposes an added layer of prototypes designers might deliver, such as interactive behaviours in the context of the form, as well as behaviours as driven by the users' actions (i.e. *interaction-driven animations*).

Many of the design tools in HCI research promise hypothetical futures, but the only way to truly know if these tools might indeed change people's practices is by becoming part of the practice as fully fledged tools with ongoing support. For example, as described early in the Soul-Body Prototyping motivation, Chapter 6, computers forever changed how graphic design was done, but for that change to happen all the software and existing infrastructure had to be in place for the technological shift to happen. Once the technology is widely available, a change in practice can indeed take place. I believe that interaction design tools are still at an early stage, given that no commercial software has remained as industry standard for more than a few years in the past decade. In that sense, the role of an interaction designer has been exclusively defined by what the currently available commercial tools can create. Today's design tools (discussed in Chapter 2) seem to converge towards large standardization of interfaces and little opportunities to diverge and customize outside of these boundaries. While there is value to base standards and generalizability, if every single interface looks and feels the same, then there is no room for custom or interesting user experiences that break away from the norm.

A question beyond the scope of this thesis, is how then, can the work produced in this thesis potentially become, or influence, part of future practices? I believe the major step will need to be for interaction design as a discipline to further mature, and gain a better sense of what it is about and what is possible. This can be achieved through (1) integrative theories that can continue to define interactive behaviours, (2) by devising new methods to enable designers to prototype, and (3) through technical explorations that help show what is possible and raise new questions that can help inform the theories. As I progressed through this thesis, my software and hardware explorations led me to find new ways for designers to envision their creations, and that led to larger questions of what is interactive behaviour and how can designers further elevate their current designs. While these tools may not become part of everyday practice in the short term, the ideas can then live and inform future systems both in research and industry.

10.6 CLOSING REMARKS

Overall, as the kinds of applications in computing continue to expand, our tools need to accordingly catch up to enable easy authoring. De-Line (Fraser et al., 2015) discusses how often times programming languages that are not at the inception of a new area struggle to gain adopters even if they are technically superior or have higher expressive match. This may also be the case in higher-level tools. In particular with prototyping tools, the challenge is the fast-moving target of all the base platforms (e.g. electronics, mobile programming platforms, web libraries, design standard) which can quickly become obsolete before the tools gain traction. Our duty as HCI researchers is

to explore different kinds of platforms and approaches to (1) stay relevant in terms of the authoring we support; and (2) repurpose existing strategies into new contexts. The explorations in interactive behaviour and smart interactive object prototyping brought forth in this thesis open potential avenues to significantly help reclaim interaction design as a discipline to build functional prototypes rather than continue a trend of focusing on simple transitions that take place on highly polished static visuals.

REFERENCES

1. Anderson, F., Grossman, T., & Fitzmaurice, G. (2017). Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, 331–342. doi: [10.1145/3126594.3126637](https://doi.org/10.1145/3126594.3126637)
2. Apitz, G., & Guimbretière, F. (2004). CrossY: A Crossing-Based Drawing Application. *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 3–12. doi: [10.1145/1029632.1029635](https://doi.org/10.1145/1029632.1029635)
3. Appert, C., & Beaudouin-Lafon, M. (2006). SwingStates: Adding State Machines to the Swing Toolkit. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 319–322. doi: [10.1145/1166253.1166302](https://doi.org/10.1145/1166253.1166302)
4. Ashbrook, D., & Starner, T. (2010). MAGIC: A Motion Gesture Design Tool. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2159–2168. doi: [10.1145/1753326.1753653](https://doi.org/10.1145/1753326.1753653)
5. Badam, S. K., & Elmquist, N. (2014). PolyChrome: A Cross-Device Framework for Collaborative Web Visualization. *Proceedings of the Ninth ACM International Conference on Interactive Tabletops and Surfaces*, 109–118. doi: [10.1145/2669485.2669518](https://doi.org/10.1145/2669485.2669518)

6. Baecker, R. M. (1969). Picture-Driven Animation. *Proceedings of the May 14-16, 1969, Spring Joint Computer Conference*, 273–288. doi: [10.1145/1476793.1476838](https://doi.org/10.1145/1476793.1476838)
7. Bailey, B. P., Konstan, J. A., & Carlis, J. V. (2001). DEMAIS: Designing Multimedia Applications with Interactive Storyboards. *Proceedings of the Ninth ACM International Conference on Multimedia*, 241–250. doi: [10.1145/500141.500179](https://doi.org/10.1145/500141.500179)
8. Ball, T., Protzenko, J., Bishop, J., Moskal, M., de Halleux, J., Braun, M., ... Riley, C. (2016). Microsoft Touch Develop and the BBC micro:bit. *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*, 637–640.
9. Ballagas, R., Ringel, M., Stone, M., & Borchers, J. (2003). iStuff: A physical user interface toolkit for ubiquitous computing environments. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 537–544. doi: [10.1145/642611.642705](https://doi.org/10.1145/642611.642705)
10. Ballendat, T., Marquardt, N., & Greenberg, S. (2010). Proxemic Interaction: Designing for a Proximity and Orientation-Aware Environment. *ACM International Conference on Interactive Tabletops and Surfaces*, 121–130. doi: [10.1145/1936652.1936676](https://doi.org/10.1145/1936652.1936676)
11. Baskinger, M., & Bardel, W. (2013). *Drawing Ideas: A Hand-drawn Approach for Better Design*. Watson-Guptill Publications.

12. Bau, O., & Mackay, W. E. (2008). OctoPocus: A Dynamic Guide for Learning Gesture-Based Command Sets. *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, 37–46. doi: [10.1145/1449715.1449724](https://doi.org/10.1145/1449715.1449724)
13. Beaudouin-Lafon, M. (2000). Instrumental Interaction: An Interaction Model for Designing Post-WIMP User Interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 446–453. doi: [10.1145/332040.332473](https://doi.org/10.1145/332040.332473)
14. Beaudouin-Lafon, M. (2004). Designing Interaction, Not Interfaces. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 15–22. doi: [10.1145/989863.989865](https://doi.org/10.1145/989863.989865)
15. Bederson, B.B., Grosjean, J., & Meyer, J. (2004). Toolkit Design for Interactive Structured Graphics. *IEEE Transactions on Software Engineering*, 30(8), 535–546. doi: [10.1109/TSE.2004.44](https://doi.org/10.1109/TSE.2004.44)
16. Bederson, Benjamin B., Hollan, J. D., Druin, A., Stewart, J., Rogers, D., & Proft, D. (1996). Local Tools: An Alternative to Tool Palettes. *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, 169–170. doi: [10.1145/237091.237116](https://doi.org/10.1145/237091.237116)
17. Bernstein, M. S., Ackerman, M. S., Chi, E. H., & Miller, R. C. (2011). The Trouble with Social Computing Systems Research. *CHI '11 Extended Abstracts on Human Factors in Computing Systems*, 389–398. doi: [10.1145/1979742.1979618](https://doi.org/10.1145/1979742.1979618)

18. Bertelsen, O. W. (2000). Design Artefacts: Towards a Design-Oriented Epistemology. *Scandinavian Journal of Information Systems*, 12(1), 2.
19. Blackburn, S. M., Garner, R., Hoffmann, C., Khang, A. M., McKinley, K. S., Bentzur, R., ... Wiedermann, B. (2006). The DaCapo Benchmarks: Java Benchmarking Development and Analysis. *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages, and Applications*, 169–190. doi: [10.1145/1167473.1167488](https://doi.org/10.1145/1167473.1167488)
20. Blackwell, A. F., Green, T. R., & Nunn, D. J. (2000). Cognitive dimensions and musical notation systems. *Proceedings of International Computer Music Conference, Berlin*.
21. Blackwell, A., & Green, T. (2007). *A Cognitive Dimensions Questionnaire*. February.
22. Block, F., Haller, M., Gellersen, H., Gutwin, C., & Billinghurst, M. (2008). VoodooSketch: Extending Interactive Surfaces with Adaptable Interface Palettes. *Proceedings of the 2nd International Conference on Tangible and Embedded Interaction*, 55–58. doi: [10.1145/1347390.1347404](https://doi.org/10.1145/1347390.1347404)
23. Bødker, S., & Grønbæk, K. (1991). Cooperative Prototyping: Users and Designers in Mutual Activity. *International Journal of Man-Machine Studies*, 34(3), 453–478. doi: [10.1016/0020-7373\(91\)90030-B](https://doi.org/10.1016/0020-7373(91)90030-B)
24. Booth, T., Stumpf, S., Bird, J., & Jones, S. (2016). Crossed Wires: Investigating the Problems of End-User Developers

- in a Physical Computing Task. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 3485–3497. doi: [10.1145/2858036.2858533](https://doi.org/10.1145/2858036.2858533)
25. Boring, S., Ledo, D., Chen, X. “Anthony,” Marquardt, N., Tang, A., & Greenberg, S. (2012). The Fat Thumb: Using the Thumb’s Contact Size for Single-Handed Mobile Interaction. *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 39–48. doi: [10.1145/2371574.2371582](https://doi.org/10.1145/2371574.2371582)
 26. Bostock, M., & Heer, J. (2009). Protovis: A Graphical Toolkit for Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 15(6), 1121–1128. doi: [10.1109/TVCG.2009.174](https://doi.org/10.1109/TVCG.2009.174)
 27. Bostock, M., Ogievetsky, V., & Heer, J. (2011). D³ Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12), 2301–2309. doi: [10.1109/TVCG.2011.185](https://doi.org/10.1109/TVCG.2011.185)
 28. Brandt, J., Guo, P. J., Lewenstein, J., & Klemmer, S. R. (2008). Opportunistic Programming: How Rapid Ideation and Prototyping Occur in Practice. *Proceedings of the 4th International Workshop on End-User Software Engineering*, 1–5. doi: [10.1145/1370847.1370848](https://doi.org/10.1145/1370847.1370848)
 29. Brudy, F., Ledo, D., Greenberg, S., & Butz, A. (2014). Is Anyone Looking? Mitigating Shoulder Surfing on Public Displays through Awareness and Protection. *Proceedings of*

The International Symposium on Pervasive Displays, 1–6. doi: [10.1145/2611009.2611028](https://doi.org/10.1145/2611009.2611028)

30. Buchanan, R. (1992). Wicked Problems in Design Thinking. *Design Issues*, 8(2), 5–21. doi: [10.2307/1511637](https://doi.org/10.2307/1511637)
31. Bardel, W. (2018). *Ubiety: On Design, Place and the Importance of Manners*. Retrieved from <https://www.youtube.com/watch?v=GEE-ZFW7PFBI&t=2526s>
32. Buxton, B. (2010). *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann.
33. Buxton, W. (1983). Lexical and Pragmatic Considerations of Input Structures. *ACM SIGGRAPH Computer Graphics*, 17(1), 31–37. doi: [10.1145/988584.988586](https://doi.org/10.1145/988584.988586)
34. Buxton, W. (1986). Chunking and Phrasing and the Design of Human-Computer Dialogues. In *Readings in Human-Computer Interaction* (pp. 494–499). Elsevier.
35. Buxton, W. (1990). A Three-State Model of Graphical Input. *Human-Computer Interaction-Interact*, 90, 449–456. Citeseer.
36. Buxton, W. (1997). Living in Augmented Reality: Ubiquitous Media and Reactive Environments. *Video Mediated Communication*, 363–384.
37. Caplan, R. (1982). *By Design: Why There Are No Locks on the Bathroom Doors in the Hotel Louis XIV, and Other Object Lessons*. Fairchild Publications.

38. Cardoso, J., & José, R. (2012). PuReWidgets: A Programming Toolkit for Interactive Public Display Applications. *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 51–60. doi: [10.1145/2305484.2305496](https://doi.org/10.1145/2305484.2305496)
39. Carroll, J. M. (2003). *HCI Models, Theories, and Frameworks: Toward a Multidisciplinary Science*. Elsevier.
40. Charmaz, K. (2014). *Constructing Grounded Theory*. SAGE.
41. Chen, X. “Anthony,” Grossman, T., Wigdor, D. J., & Fitzmaurice, G. (2014). Duet: Exploring Joint Interactions on a Smart Phone and a Smart Watch. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 159–168. doi: [10.1145/2556288.2556955](https://doi.org/10.1145/2556288.2556955)
42. Chen, X. “Anthony,” Marquardt, N., Tang, A., Boring, S., & Greenberg, S. (2012). Extending a Mobile Device’s Interaction Space Through Body-Centric Interaction. *Proceedings of the 14th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 151–160. doi: [10.1145/2371574.2371599](https://doi.org/10.1145/2371574.2371599)
43. Cheung, V., & Girouard, A. (2019). Tangible Around-Device Interaction Using Rotatory Gestures with a Magnetic Ring. *Proceedings of the 21st International Conference on Human-Computer Interaction with Mobile Devices and Services*, 1–8. doi: [10.1145/3338286.3340137](https://doi.org/10.1145/3338286.3340137)
44. Chi, P.-Y. (Peggy), & Li, Y. (2015). Weave: Scripting Cross-Device Wearable Interaction. *Proceedings of the 33rd Annual*

ACM Conference on Human Factors in Computing Systems,
3923–3932. doi: [10.1145/2702123.2702451](https://doi.org/10.1145/2702123.2702451)

45. Chimero, F. (2012). *The Shape of Design*.
46. Collective, B. M., & Shaw, D. (2012). Makey Makey: Improvising Tangible and Nature-Based User Interfaces. *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, 367–370. doi: [10.1145/2148131.2148219](https://doi.org/10.1145/2148131.2148219)
47. Cooper, A., Reimann, R., Cronin, D., & Noessel, C. (2014). *About Face: The Essentials of Interaction Design*. John Wiley & Sons.
48. Cooperstock, J. R., Fels, S. S., Buxton, W., & Smith, K. C. (1997). Reactive Environments. *Communications of the ACM*, 40(9), 65–73. doi: [10.1145/260750.260774](https://doi.org/10.1145/260750.260774)
49. Cross, N. (1982). Designerly ways of knowing. *Design Studies*, 3(4), 221–227. doi: [10.1016/0142-694X\(82\)90040-0](https://doi.org/10.1016/0142-694X(82)90040-0)
50. Cross, N. (1997). Descriptive Models of Creative Design: Application to an Example. *Design Studies*, 18(4), 427–440. doi: [10.1016/S0142-694X\(97\)00010-0](https://doi.org/10.1016/S0142-694X(97)00010-0)
51. Cross, N. (1999). Design Research: A Disciplined Conversation. *Design Issues*, 15(2), 5–10. doi: [10.2307/1511837](https://doi.org/10.2307/1511837)
52. Cross, N. (2011). *Design Thinking: Understanding How Designers Think and Work*. Berg.

53. Dalsgaard, P. (n.d.). Instruments of Inquiry: Understanding the Nature and Role of Tools in Design. *International Journal of Design*. Retrieved from <http://index.ijdesign.org/index.php/IJDesign/article/view/2275>
54. Danis, C., & Boies, S. (2000). Using a Technique from Graphic Designers to Develop Innovative System Designs. *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, 20–26. doi: [10.1145/347642.347657](https://doi.org/10.1145/347642.347657)
55. de Alwis, B., Gutwin, C., & Greenberg, S. (2009). GT/SD: Performance and Simplicity in a Groupware Toolkit. *Proceedings of the 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 265–274. doi: [10.1145/1570433.1570483](https://doi.org/10.1145/1570433.1570483)
56. de Sá, M., Carriço, L., Duarte, L., & Reis, T. (2008). A Mixed-Fidelity Prototyping Tool for Mobile Devices. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 225–232. doi: [10.1145/1385569.1385606](https://doi.org/10.1145/1385569.1385606)
57. Dey, A. K., Hamid, R., Beckmann, C., Li, I., & Hsu, D. (2004). A CAPpella: Programming by Demonstration of Context-Aware Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 33–40. doi: [10.1145/985692.985697](https://doi.org/10.1145/985692.985697)
58. Dey, A. K., & Newberger, A. (2009). Support for Context-Aware Intelligibility and Control. *Proceedings of the SIGCHI*

- Conference on Human Factors in Computing Systems*, 859–868.
doi: [10.1145/1518701.1518832](https://doi.org/10.1145/1518701.1518832)
59. Dix, A. J., Finlay, J., Abowd, G. D., & Beale, R. (2003). *Human-Computer Interaction*. Pearson Education.
 60. Dixon, M., & Fogarty, J. (2010). Prefab: Implementing Advanced Behaviors Using Pixel-Based Reverse Engineering of Interface Structure. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1525–1534. doi: [10.1145/1753326.1753554](https://doi.org/10.1145/1753326.1753554)
 61. Dragicevic, P., & Fekete, J.-D. (2004). Support for Input Adaptability in the Icon Toolkit. *Proceedings of the 6th International Conference on Multimodal Interfaces*, 212–219. doi: [10.1145/1027933.1027969](https://doi.org/10.1145/1027933.1027969)
 62. Dumas, J. S., Dumas, J. S., & Redish, J. (1999). *A Practical Guide to Usability Testing*. Intellect Books.
 63. Edwards, W. K., Newman, M. W., & Poole, E. S. (2010). The infrastructure problem in HCI. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 423–432. doi: [10.1145/1753326.1753390](https://doi.org/10.1145/1753326.1753390)
 64. Engelbart, D. C. (1968). The Mother of All Demos. *Fall Joint Computer Conference, San Francisco, Dec, 9*.
 65. Finzer, W. F., & Gould, L. (1993). Rehearsal world: Programming by rehearsal. In *Watch what I do: Programming by demonstration* (pp. 79–100).

66. Fitzmaurice, G., & Buxton, W. (1994). The Chameleon: Spatially Aware Palmtop Computers. *Conference Companion on Human Factors in Computing Systems*, 451–452. doi: [10.1145/259963.260460](https://doi.org/10.1145/259963.260460)
67. Floyd, C. (1984). A Systematic Look at Prototyping. In R. Budde, K. Kuhlenkamp, L. Mathiassen, & H. Züllighoven (Eds.), *Approaches to Prototyping* (pp. 1–18). doi: [10.1007/978-3-642-69796-8_1](https://doi.org/10.1007/978-3-642-69796-8_1)
68. Fogarty, J. (2017). Code and Contribution in Interactive Systems Research. *Workshop on HCI. Tools at CHI*.
69. Foley, J. D., Van, F. D., Dam, A. V., Feiner, S. K., Hughes, J. F., Angel, E., & Hughes, J. (1990). *Computer Graphics: Principles and Practice*. Addison-Wesley Professional.
70. Follmer, S., Carr, D., Lovell, E., & Ishii, H. (2010). Copy-CAD: Remixing Physical Objects with Copy and Paste from the Real World. *Adjunct Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, 381–382. doi: [10.1145/1866218.1866230](https://doi.org/10.1145/1866218.1866230)
71. Follmer, S., & Ishii, H. (2012). KidCAD: Digitally Remixing Toys through Tangible Tools. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2401–2410. doi: [10.1145/2207676.2208403](https://doi.org/10.1145/2207676.2208403)
72. Fraser, S. D., Bak, L., DeLine, R., Feamster, N., Kuper, L., Lopes, C. V., & Wu, P. (2015). The Future of Programming Languages and Programmers. *Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems*,

- Programming, Languages and Applications: Software for Humanity*, 63–66. doi: [10.1145/2814189.2818719](https://doi.org/10.1145/2814189.2818719)
73. Gaines, B. R. (1991). Modeling and Forecasting the Information Sciences. *Information Sciences*, 57–58, 3–22. doi: [10.1016/0020-0255\(91\)90066-4](https://doi.org/10.1016/0020-0255(91)90066-4)
 74. Genest, A. M., Gutwin, C., Tang, A., Kalyn, M., & Ivkovic, Z. (2013). KinectArms: A Toolkit for Capturing and Displaying Arm Embodiments in Distributed Tabletop Groupware. *Proceedings of the 2013 Conference on Computer Supported Cooperative Work*, 157–166. doi: [10.1145/2441776.2441796](https://doi.org/10.1145/2441776.2441796)
 75. Gerber, E., & Carroll, M. (2012). The Psychological Experience of Prototyping. *Design Studies*, 33(1), 64–84. doi: [10.1016/j.destud.2011.06.005](https://doi.org/10.1016/j.destud.2011.06.005)
 76. Gero, J. S. (1990). Design Prototypes: A Knowledge Representation Schema for Design. *AI Magazine*, 11(4), 26–26. doi: [10.1609/aimag.v11i4.854](https://doi.org/10.1609/aimag.v11i4.854)
 77. Goel, V., & Pirolli, P. (1992). The Structure of Design Problem Spaces. *Cognitive Science*, 16(3), 395–429. doi: [10.1207/s15516709cog1603_3](https://doi.org/10.1207/s15516709cog1603_3)
 78. Gould, J. D., Conti, J., & Hovanyecz, T. (1983). Composing Letters with a Simulated Listening Typewriter. *Communications of the ACM*, 26(4), 295–308. doi: [10.1145/2163.358100](https://doi.org/10.1145/2163.358100)

79. Greenberg, S. (2001). Context as a Dynamic Construct. *Human-Computer Interaction*, 16(2–4), 257–268. doi: [10.1207/S15327051HCI16234_09](https://doi.org/10.1207/S15327051HCI16234_09)
80. Greenberg, S. (2007). Toolkits and Interface Creativity. *Multimedia Tools and Applications*, 32(2), 139–159. doi: [10.1007/s11042-006-0062-y](https://doi.org/10.1007/s11042-006-0062-y)
81. Greenberg, S., & Buxton, B. (2008). Usability Evaluation Considered Harmful (some of the Time). *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 111–120. doi: [10.1145/1357054.1357074](https://doi.org/10.1145/1357054.1357074)
82. Greenberg, S., Carpendale, S., Marquardt, N., & Buxton, B. (2011). *Sketching User Experiences: The Workbook*. Elsevier.
83. Greenberg, S., & Fitchett, C. (2001). Phidgets: Easy Development of Physical Interfaces through Physical Widgets. *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, 209–218. doi: [10.1145/502348.502388](https://doi.org/10.1145/502348.502388)
84. Greenberg, S., & Roseman, M. (1996). GroupWeb: A WWW Browser as Real Time GroupWare. *Conference Companion on Human Factors in Computing Systems*, 271–272. doi: [10.1145/257089.257317](https://doi.org/10.1145/257089.257317)
85. Grigoreanu, V., Fernandez, R., Inkpen, K., & Robertson, G. (2009). What Designers Want: Needs of Interactive Application Designers. *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 139–146. doi: [10.1109/VLHCC.2009.5295277](https://doi.org/10.1109/VLHCC.2009.5295277)

86. Grosse-Puppendahl, T., Berghofer, Y., Braun, A., Wimmer, R., & Kuijper, A. (2013). Opencapsense: A Rapid Prototyping Toolkit for Pervasive Interaction Using Capacitive Sensing. *2013 IEEE International Conference on Pervasive Computing and Communications (PerCom)*, 152–159. doi: [10.1109/PerCom.2013.6526726](https://doi.org/10.1109/PerCom.2013.6526726)
87. Grossman, T., & Balakrishnan, R. (2005). The Bubble Cursor: Enhancing Target Acquisition by Dynamic Resizing of the Cursor's Activation Area. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 281–290. doi: [10.1145/1054972.1055012](https://doi.org/10.1145/1054972.1055012)
88. Halbert, D. C. (1984). *Programming by Example* (PhD Thesis). University of California, Berkeley.
89. Hancock, C. M. (2003). *Real-Time Programming and the Big Ideas of Computational Literacy* (Thesis, Massachusetts Institute of Technology). Retrieved from <https://dspace.mit.edu/handle/1721.1/61549>
90. Hansen, T. E., Hourcade, J. P., Virbel, M., Patali, S., & Serra, T. (2009). PyMT: A Post-WIMP Multi-Touch User Interface Toolkit. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 17–24. doi: [10.1145/1731903.1731907](https://doi.org/10.1145/1731903.1731907)
91. Harrison, C., Schwarz, J., & Hudson, S. E. (2011). Tapsense: Enhancing Finger Interaction on Touch Surfaces.

- Proceedings of the 24th Annual Acm Symposium on User Interface Software and Technology*, 627–636. doi: [10.1145/2047196.2047279](https://doi.org/10.1145/2047196.2047279)
92. Hartmann, Bjorn. (2009). *Gaining Design Insight through Interaction Prototyping Tools* (Stanford University). Retrieved from <https://people.eecs.berkeley.edu/~bjoern/dissertation/hartmann-diss.pdf>
 93. Hartmann, Björn, Abdulla, L., Mittal, M., & Klemmer, S. R. (2007a). Authoring Sensor-Based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 145–154. doi: [10.1145/1240624.1240646](https://doi.org/10.1145/1240624.1240646)
 94. Hartmann, Björn, Abdulla, L., Mittal, M., & Klemmer, S. R. (2007b). Authoring Sensor-Based Interactions by Demonstration with Direct Manipulation and Pattern Recognition. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 145–154. doi: [10.1145/1240624.1240646](https://doi.org/10.1145/1240624.1240646)
 95. Hartmann, Björn, Klemmer, S. R., Bernstein, M., Abdulla, L., Burr, B., Robinson-Mosher, A., & Gee, J. (2006). Reflective Physical Prototyping Through Integrated Design, Test, and Analysis. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 299–308. doi: [10.1145/1166253.1166300](https://doi.org/10.1145/1166253.1166300)
 96. Heer, J., Card, S. K., & Landay, J. A. (2005). prefuse: A toolkit for interactive information visualization. *Proceedings*

of the SIGCHI Conference on Human Factors in Computing Systems, 421–430. doi: [10.1145/1054972.1055031](https://doi.org/10.1145/1054972.1055031)

97. Hempel, B., & Chugh, R. (2016). Semi-Automated SVG Programming via Direct Manipulation. *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, 379–390. doi: [10.1145/2984511.2984575](https://doi.org/10.1145/2984511.2984575)
98. Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design Science in Information Systems Research. *MIS Quarterly*, 28(1), 75–105. doi: [10.2307/25148625](https://doi.org/10.2307/25148625)
99. Hewett, T. T., Baecker, R., Card, S., Carey, T., Gasen, J., Mantei, M., ... Verplank, W. (1992). *ACM SIGCHI Curriculum for Human-Computer Interaction*. New York, NY, USA: Association for Computing Machinery.
100. Hill, J., & Gutwin, C. (2004). The MAUI Toolkit: Groupware Widgets for Group Awareness. *Computer Supported Cooperative Work (CSCW)*, 13(5), 539–571. doi: [10.1007/s10606-004-5063-7](https://doi.org/10.1007/s10606-004-5063-7)
101. Hinckley, K. (2003). Synchronous Gestures for Multiple Persons and Computers. *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, 149–158. doi: [10.1145/964696.964713](https://doi.org/10.1145/964696.964713)
102. Hinckley, K., Jacob, R. J., Ware, C., Wobbrock, J. O., & Wigdor, D. (2014). *Input/Output Devices and Interaction Techniques*.

103. Hinckley, K., Pahud, M., Benko, H., Irani, P., Guimbretière, F., Gavriliu, M., ... Wilson, A. (2014). Sensing Techniques for Tablet+stylus Interaction. *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, 605–614. doi: [10.1145/2642918.2647379](https://doi.org/10.1145/2642918.2647379)
104. Hinckley, K., Pierce, J., Sinclair, M., & Horvitz, E. (2000). Sensing Techniques for Mobile Interaction. *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, 91–100. doi: [10.1145/354401.354417](https://doi.org/10.1145/354401.354417)
105. Hinckley, K., & Song, H. (2011). Sensor Synesthesia: Touch in Motion, and Motion in Touch. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 801–810. doi: [10.1145/1978942.1979059](https://doi.org/10.1145/1978942.1979059)
106. Hodges, S., Taylor, S., Villar, N., Scott, J., & Helmes, J. (2013). Exploring Physical Prototyping Techniques for Functional Devices Using .NET Gadgeteer. *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, 271–274. doi: [10.1145/2460625.2460670](https://doi.org/10.1145/2460625.2460670)
107. Holmquist, L. E. (2005). Prototyping: Generating Ideas or Cargo Cult Designs? *Interactions*, 12(2), 48–54. doi: [10.1145/1052438.1052465](https://doi.org/10.1145/1052438.1052465)
108. Holmquist, L. E., Mattern, F., Schiele, B., Alahuhta, P., Beigl, M., & Gellersen, H.-W. (2001). Smart-Its Friends: A Technique for Users to Easily Establish Connections between Smart Artefacts. In G. D. Abowd, B. Brumitt, & S.

- Shafer (Eds.), *Ubicomp 2001: Ubiquitous Computing* (pp. 116–122). doi: [10.1007/3-540-45427-6_10](https://doi.org/10.1007/3-540-45427-6_10)
109. Hong, J. I., & Landay, J. A. (2006). SATIN: A Toolkit for Informal Ink-Based Applications. *ACM SIGGRAPH 2006 Courses*, 7–es. doi: [10.1145/1185657.1185768](https://doi.org/10.1145/1185657.1185768)
110. Horak, T., Badam, S. K., Elmqvist, N., & Dachselt, R. (2018). When David Meets Goliath: Combining Smartwatches with a Large Vertical Display for Visual Data Exploration. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–13. doi: [10.1145/3173574.3173593](https://doi.org/10.1145/3173574.3173593)
111. Hornbæk, K., & Oulasvirta, A. (2017). What Is Interaction? *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 5040–5052. doi: [10.1145/3025453.3025765](https://doi.org/10.1145/3025453.3025765)
112. Houben, S., Golsteijn, C., Gallacher, S., Johnson, R., Bakker, S., Marquardt, N., ... Rogers, Y. (2016). Physikit: Data Engagement Through Physical Ambient Visualizations in the Home. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 1608–1619. doi: [10.1145/2858036.2858059](https://doi.org/10.1145/2858036.2858059)
113. Houben, S., & Marquardt, N. (2015). WatchConnect: A Toolkit for Prototyping Smartwatch-Centric Cross-Device Applications. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 1247–1256. doi: [10.1145/2702123.2702215](https://doi.org/10.1145/2702123.2702215)

114. Houde, S., & Hill, C. (1997). Chapter 16—What do Prototypes Prototype? In M. G. Helander, T. K. Landauer, & P. V. Prabhu (Eds.), *Handbook of Human-Computer Interaction (Second Edition)* (pp. 367–381). doi: [10.1016/B978-044481862-1.50082-0](https://doi.org/10.1016/B978-044481862-1.50082-0)
115. Hudson, S. E., & Mankoff, J. (2006). Rapid Construction of Functioning Physical Interfaces from Cardboard, Thumbtacks, Tin Foil and Masking Tape. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 289–298. doi: [10.1145/1166253.1166299](https://doi.org/10.1145/1166253.1166299)
116. Hudson, S. E., & Mankoff, J. (2014). Concepts, Values, and Methods for Technical Human–Computer Interaction Research. In J. S. Olson & W. A. Kellogg (Eds.), *Ways of Knowing in HCI* (pp. 69–93). doi: [10.1007/978-1-4939-0378-8_4](https://doi.org/10.1007/978-1-4939-0378-8_4)
117. Hudson, S. E., Mankoff, J., & Smith, I. (2005). Extensible Input Handling in the SubArctic Toolkit. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 381–390. doi: [10.1145/1054972.1055025](https://doi.org/10.1145/1054972.1055025)
118. Huot, S., Dumas, C., Dragicevic, P., Fekete, J.-D., & Hégron, G. (2004). The MaggLite post-WIMP toolkit: Draw it, connect it and run it. *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 257–266. doi: [10.1145/1029632.1029677](https://doi.org/10.1145/1029632.1029677)
119. Hwang, S., Ahn, M., & Wohn, K. (2013). MagGetz: Customizable Passive Tangible Controllers on and Around Conventional Mobile Devices. *Proceedings of the 26th Annual ACM*

- Symposium on User Interface Software and Technology*, 411–416. doi: [10.1145/2501988.2501991](https://doi.org/10.1145/2501988.2501991)
120. Hwang, S., Bianchi, A., Ahn, M., & Wohn, K. (2013). MagPen: Magnetically Driven Pen Interactions on and Around Conventional Smartphones. *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 412–415. doi: [10.1145/2493190.2493194](https://doi.org/10.1145/2493190.2493194)
 121. Intelligibility and Accountability: Human Considerations in Context-Aware Systems: Human–Computer Interaction: Vol 16, No 2-4. (n.d.). Retrieved July 7, 2020, from https://www.tandfonline.com/doi/abs/10.1207/S15327051HCI16234_05?casa_token=dF_tbMp-PhF0AAAAAA:EJJ5JB4U2v6z6omBTz-lIxxZ8l1jf_qTGIKdfQEpstZlPJyUazpdsg-WUyPo1_o5nC5BmyNbTw_k
 122. Janlert, L.-E., & Stolterman, E. (2017). The Meaning of Interactivity—Some Proposals for Definitions and Measures. *Human–Computer Interaction*, 32(3), 103–138. doi: [10.1080/07370024.2016.1226139](https://doi.org/10.1080/07370024.2016.1226139)
 123. Johnson, J., Roberts, T. L., Verplank, W., Smith, D. C., Irby, C. H., Beard, M., & Mackey, K. (1989). The Xerox Star: A Retrospective. *Computer*, 22(9), 11–26. doi: [10.1109/2.35211](https://doi.org/10.1109/2.35211)
 124. Johnston, O., & Thomas, F. (1981). *The Illusion of Life: Disney Animation*. Disney Editions New York.

125. Ju, W., & Leifer, L. (2008). The Design of Implicit Interactions: Making Interactive Systems Less Obnoxious. *Design Issues*, 24(3), 72–84. doi: [10.1162/desi.2008.24.3.72](https://doi.org/10.1162/desi.2008.24.3.72)
126. Kaltenbrunner, M., & Bencina, R. (2007). ReacTIVision: A Computer-Vision Framework for Table-Based Tangible Interaction. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 69–74. doi: [10.1145/1226969.1226983](https://doi.org/10.1145/1226969.1226983)
127. Kato, J., Sakamoto, D., & Igarashi, T. (2012). Phybots: A Toolkit for Making Robotic Things. *Proceedings of the Designing Interactive Systems Conference*, 248–257. doi: [10.1145/2317956.2317996](https://doi.org/10.1145/2317956.2317996)
128. Kato, K., & Miyashita, H. (2014). Extension Sticker: A Method for Transferring External Touch Input Using a Striped Pattern Sticker. *Proceedings of the Adjunct Publication of the 27th Annual ACM Symposium on User Interface Software and Technology*, 59–60. doi: [10.1145/2658779.2668032](https://doi.org/10.1145/2658779.2668032)
129. Kaufmann, B., & Buechley, L. (2010). Amarino: A Toolkit for the Rapid Prototyping of Mobile Ubiquitous Computing. *Proceedings of the 12th International Conference on Human Computer Interaction with Mobile Devices and Services*, 291–298. doi: [10.1145/1851600.1851652](https://doi.org/10.1145/1851600.1851652)
130. Kaye, J. “Jofish.” (2007). Evaluating experience-focused HCI. *CHI ’07 Extended Abstracts on Human Factors in Computing Systems*, 1661–1664. doi: [10.1145/1240866.1240877](https://doi.org/10.1145/1240866.1240877)

131. Kazi, R. H., Chevalier, F., Grossman, T., & Fitzmaurice, G. (2014). Kitty: Sketching Dynamic and Interactive Illustrations. *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, 395–405. doi: [10.1145/2642918.2647375](https://doi.org/10.1145/2642918.2647375)
132. Kazi, R. H., Grossman, T., Umetani, N., & Fitzmaurice, G. (2016). Motion Amplifiers: Sketching Dynamic Illustrations Using the Principles of 2D Animation. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 4599–4609. doi: [10.1145/2858036.2858386](https://doi.org/10.1145/2858036.2858386)
133. Kelley, J. F. (1983). An Empirical Methodology for Writing User-Friendly Natural Language Computer Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 193–196. doi: [10.1145/800045.801609](https://doi.org/10.1145/800045.801609)
134. Kirton, T., Boring, S., Baur, D., MacDonald, L., & Carpendale, S. (2013). C4: A Creative-Coding API for Media, Interaction and Animation. *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, 279–286. doi: [10.1145/2460625.2460672](https://doi.org/10.1145/2460625.2460672)
135. Klemmer, S. R., Li, J., Lin, J., & Landay, J. A. (2004). Paper-Mache: Toolkit Support for Tangible Input. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 399–406. doi: [10.1145/985692.985743](https://doi.org/10.1145/985692.985743)
136. Ko, A., Myers, B. A., & Aung, H. H. (2004). Six Learning Barriers in End-User Programming Systems. *2004 IEEE*

- Symposium on Visual Languages - Human Centric Computing*, 199–206. doi: [10.1109/VLHCC.2004.47](https://doi.org/10.1109/VLHCC.2004.47)
137. Krosnick, R., Lee, S. W., Laseck, W. S., & Onev, S. (2018). Espresso: Building Responsive Interfaces with Keyframes. *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 39–47. doi: [10.1109/VLHCC.2018.8506516](https://doi.org/10.1109/VLHCC.2018.8506516)
 138. Landay, J. A. (1996). SILK: Sketching Interfaces Like Krazy. *Conference Companion on Human Factors in Computing Systems*, 398–399. doi: [10.1145/257089.257396](https://doi.org/10.1145/257089.257396)
 139. Laput, G., Brockmeyer, E., Hudson, S. E., & Harrison, C. (2015). Acoustruments: Passive, Acoustically-Driven, Interactive Controls for Handheld Devices. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2161–2170. doi: [10.1145/2702123.2702414](https://doi.org/10.1145/2702123.2702414)
 140. Laseau, P. (1982). *Graphic Thinking for Architects and Designers*. Elsevier.
 141. Lazar, J., Feng, J. H., & Hochheiser, H. (2017). *Research Methods in Human-Computer Interaction*. Morgan Kaufmann.
 142. Ledo, D., Greenberg, S., Marquardt, N., & Boring, S. (2015). Proxemic-Aware Controls: Designing Remote Controls for Ubiquitous Computing Ecologies. *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 187–198. doi: [10.1145/2785830.2785871](https://doi.org/10.1145/2785830.2785871)

143. Ledo, D., Nacenta, M. A., Marquardt, N., Boring, S., & Greenberg, S. (2012). The Haptictouch Toolkit: Enabling Exploration of Haptic Interactions. *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, 115–122. doi: [10.1145/2148131.2148157](https://doi.org/10.1145/2148131.2148157)
144. Lee, J. C. (2008). *Wii remote hacks | Johnny Lee*. Retrieved from <https://www.youtube.com/watch?v=QgKCrGvShZs>
145. Lee, J. C., Avrahami, D., Hudson, S. E., Forlizzi, J., Dietz, P. H., & Leigh, D. (2004). The Calder Toolkit: Wired and Wireless Components for Rapidly Prototyping Interactive Devices. *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, 167–175. doi: [10.1145/1013115.1013139](https://doi.org/10.1145/1013115.1013139)
146. Leiva, G. (2018). *Interactive Prototyping of Interactions: From Throwaway Prototypes to Takeaway Prototyping* (Phdthesis, Université Paris Saclay (COmUE)). Retrieved from <https://tel.archives-ouvertes.fr/tel-02122823>
147. Leiva, G., & Beaudouin-Lafon, M. (2018). Montage: A Video Prototyping System to Reduce Re-Shooting and Increase Re-Usability. *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, 675–682. doi: [10.1145/3242587.3242613](https://doi.org/10.1145/3242587.3242613)
148. Li, T. J.-J., Azaria, A., & Myers, B. A. (2017). SUGILITE: Creating Multimodal Smartphone Automation by Demonstration. *Proceedings of the 2017 CHI Conference on Human*

- Factors in Computing Systems*, 6038–6049. doi: [10.1145/3025453.3025483](https://doi.org/10.1145/3025453.3025483)
149. Li, Y., & Landay, J. A. (2005). Informal Prototyping of Continuous Graphical Interactions by Demonstration. *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, 221–230. doi: [10.1145/1095034.1095071](https://doi.org/10.1145/1095034.1095071)
 150. Lichter, H., Schneider-Hufschmidt, M., & Zullighoven, H. (1994). Prototyping in industrial software projects-bridging the gap between theory and practice. *IEEE Transactions on Software Engineering*, 20(11), 825–832. doi: [10.1109/32.368126](https://doi.org/10.1109/32.368126)
 151. Lim, B. Y., & Dey, A. K. (2010). Toolkit to Support Intelligibility in Context-Aware Applications. *Proceedings of the 12th ACM International Conference on Ubiquitous Computing*, 13–22. doi: [10.1145/1864349.1864353](https://doi.org/10.1145/1864349.1864353)
 152. Lim, Y.-K., Stolterman, E., & Tenenberg, J. (2008). The Anatomy of Prototypes: Prototypes as Filters, Prototypes as Manifestations of Design Ideas. *ACM Trans. Comput.-Hum. Interact.*, 15(2). doi: [10.1145/1375761.1375762](https://doi.org/10.1145/1375761.1375762)
 153. Lin, J., & Landay, J. A. (2008). Employing Patterns and Layers for Early-Stage Design and Prototyping of Cross-Device User Interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1313–1322. doi: [10.1145/1357054.1357260](https://doi.org/10.1145/1357054.1357260)

154. Lin, J., Newman, M. W., Hong, J. I., & Landay, J. A. (2000). DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 510–517. doi: [10.1145/332040.332486](https://doi.org/10.1145/332040.332486)
155. Lindell, R. (2014). Crafting Interaction: The Epistemology of Modern Programming. *Personal and Ubiquitous Computing*, 18(3), 613–624.
156. Logan, B., & Smithers, T. (1993). Chapter 7—Creativity and Design as Exploration. In *Modeling Creativity and Knowledge-Based Creative Design*. Lawrence Erlbaum Associates, Inc.
157. Lopes, P., Jota, R., & Jorge, J. A. (2011). Augmenting Touch Interaction Through Acoustic Sensing. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 53–56. doi: [10.1145/2076354.2076364](https://doi.org/10.1145/2076354.2076364)
158. Löwgren, J., & Stolterman, E. (2004). *Thoughtful Interaction Design: A Design Perspective on Information Technology*. MIT Press.
159. MacIntyre, B., Gandy, M., Dow, S., & Bolter, J. D. (2004). DART: A Toolkit for Rapid Design Exploration of Augmented Reality Experiences. *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 197–206. doi: [10.1145/1029632.1029669](https://doi.org/10.1145/1029632.1029669)
160. Mackay, W. E. (1988). Video Prototyping: A Technique for Developing Hypermedia Systems. *Proceedings of the 1988*

- CHI Conference on Human Factors in Computing Systems, 5, 1–3. Citeseer.*
161. Mackinlay, J., Card, S. K., & Robertson, G. G. (1990). A Semantic Analysis of the Design Space of Input Devices. *Human-Computer Interaction, 5*(2–3), 145–190. doi: [10.1080/07370024.1990.9667153](https://doi.org/10.1080/07370024.1990.9667153)
 162. Maloney, J. H., & Smith, R. B. (1995). Directness and Liveness in the Morphic User Interface Construction Environment. *Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology*, 21–28. doi: [10.1145/215585.215636](https://doi.org/10.1145/215585.215636)
 163. Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education, 10*(4), 16:1–16:15. doi: [10.1145/1868358.1868363](https://doi.org/10.1145/1868358.1868363)
 164. Mankoff, J. (2000). Providing Integrated Toolkit-Level Support for Ambiguity in Recognition-Based Interfaces. *CHI '00 Extended Abstracts on Human Factors in Computing Systems, 77–78*. doi: [10.1145/633292.633339](https://doi.org/10.1145/633292.633339)
 165. Marco, J., Cerezo, E., & Baldassarri, S. (2012). ToyVision: A Toolkit for Prototyping Tabletop Tangible Games. *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, 71–80*. doi: [10.1145/2305484.2305498](https://doi.org/10.1145/2305484.2305498)

166. Marquardt, N., Ballendat, T., Boring, S., Greenberg, S., & Hinckley, K. (2012). Gradual Engagement: Facilitating Information Exchange Between Digital Devices as a Function of Proximity. *Proceedings of the 2012 ACM International Conference on Interactive Tabletops and Surfaces*, 31–40. doi: [10.1145/2396636.2396642](https://doi.org/10.1145/2396636.2396642)
167. Marquardt, N., Diaz-Marino, R., Boring, S., & Greenberg, S. (2011). The Proximity Toolkit: Prototyping Proxemic Interactions in Ubiquitous Computing Ecologies. *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 315–326. doi: [10.1145/2047196.2047238](https://doi.org/10.1145/2047196.2047238)
168. Marquardt, N., & Greenberg, S. (2007). Distributed Physical Interfaces with Shared Phidgets. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 13–20. doi: [10.1145/1226969.1226973](https://doi.org/10.1145/1226969.1226973)
169. Marquardt, N., Houben, S., Beaudouin-Lafon, M., & Wilson, A. D. (2017). HCITools: Strategies and Best Practices for Designing, Evaluating and Sharing Technical HCI Toolkits. *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, 624–627. doi: [10.1145/3027063.3027073](https://doi.org/10.1145/3027063.3027073)
170. Marquardt, N., Kiemer, J., Ledo, D., Boring, S., & Greenberg, S. (2011). Designing User-, Hand-, and Handpart-Aware Tabletop Interactions with the TouchID Toolkit. *Proceedings of the ACM International Conference on Interactive Tabletops and Surfaces*, 21–30. doi: [10.1145/2076354.2076358](https://doi.org/10.1145/2076354.2076358)

171. Matthews, T., Dey, A. K., Mankoff, J., Carter, S., & Rattenbury, T. (2004). A toolkit for managing user attention in peripheral displays. *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, 247–256. doi: [10.1145/1029632.1029676](https://doi.org/10.1145/1029632.1029676)
172. Maudet, N., Leiva, G., Beaudouin-Lafon, M., & Mackay, W. (2017). Design Breakdowns: Designer-Developer Gaps in Representing and Interpreting Interactive Systems. *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, 630–641. doi: [10.1145/2998181.2998190](https://doi.org/10.1145/2998181.2998190)
173. McGrath, J. E. (1995). Methodology Matters: Doing Research in the Behavioral and Social Sciences. In RONALD M. Baecker, J. Grudin, W. A. S. Buxton, & S. Greenberg (Eds.), *Readings in Human-Computer Interaction* (pp. 152–169). doi: [10.1016/B978-0-08-051574-8.50019-4](https://doi.org/10.1016/B978-0-08-051574-8.50019-4)
174. Meskens, J., Luyten, K., & Coninx, K. (2009). Shortening User Interface Design Iterations Through Realtime Visualisation of Design Actions on the Target Device. *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)*, 132–135. doi: [10.1109/VLHCC.2009.5295281](https://doi.org/10.1109/VLHCC.2009.5295281)
175. Meskens, J., Luyten, K., & Coninx, K. (2010). D-Macs: Building Multi-Device User Interfaces by Demonstrating, Sharing and Replaying Design Actions. *Proceedings of the*

23nd Annual ACM Symposium on User Interface Software and Technology, 129–138. doi: [10.1145/1866029.1866051](https://doi.org/10.1145/1866029.1866051)

176. Meskens, J., Vermeulen, J., Luyten, K., & Coninx, K. (2008). Gummy for Multi-Platform User Interface Designs: Shape Me, Multiply Me, Fix Me, Use Me. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 233–240. doi: [10.1145/1385569.1385607](https://doi.org/10.1145/1385569.1385607)
177. Moggridge, B. (2007). *Designing Interactions*. MIT Press.
178. Moscovich, T. (2009). Contact Area Interaction with Sliding Widgets. *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, 13–22. doi: [10.1145/1622176.1622181](https://doi.org/10.1145/1622176.1622181)
179. Myers, B. A. (1998). Scripting Graphical Applications by Demonstration. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 534–541. doi: [10.1145/274644.274716](https://doi.org/10.1145/274644.274716)
180. Myers, B. A. (2002). Mobile Devices for Control. In F. Paternò (Ed.), *Human Computer Interaction with Mobile Devices* (pp. 1–8). doi: [10.1007/3-540-45756-9_1](https://doi.org/10.1007/3-540-45756-9_1)
181. Myers, B. A., & Buxton, W. (1986). Creating Highly-Interactive and Graphical User Interfaces by Demonstration. *SIGGRAPH Comput. Graph.*, 20(4), 249–258. doi: [10.1145/15886.15914](https://doi.org/10.1145/15886.15914)

182. Myers, B. A., Pane, J. F., & Ko, A. (2004). Natural Programming Languages and Environments. *Communications of the ACM*, 47(9), 47–52. doi: [10.1145/1015864.1015888](https://doi.org/10.1145/1015864.1015888)
183. Myers, B. A., Peck, C. H., Nichols, J., Kong, D., & Miller, R. (2001). Interacting at a Distance Using Semantic Snarfing. In G. D. Abowd, B. Brumitt, & S. Shafer (Eds.), *Ubicomp 2001: Ubiquitous Computing* (pp. 305–314). doi: [10.1007/3-540-45427-6_26](https://doi.org/10.1007/3-540-45427-6_26)
184. Myers, B., Hudson, S. E., & Pausch, R. (2000). Past, Present, and Future of User Interface Software Tools. *ACM Transactions on Computer-Human Interaction*, 7(1), 3–28. doi: [10.1145/344949.344959](https://doi.org/10.1145/344949.344959)
185. Myers, B., Park, S. Y., Nakano, Y., Mueller, G., & Ko, A. (2008). How Designers Design and Program Interactive Behaviors. *2008 IEEE Symposium on Visual Languages and Human-Centric Computing*, 177–184. doi: [10.1109/VLHCC.2008.4639081](https://doi.org/10.1109/VLHCC.2008.4639081)
186. Nardi, B. A., O’Day, V., & O’Day, V. L. (1999). *Information Ecologies: Using Technology with Heart*. MIT Press.
187. Nebeling, M. (2017). Playing the Tricky Game of Toolkits Research. *Workshop on HCI Tools at CHI*.
188. Nebeling, M., Mintsi, T., Husmann, M., & Norrie, M. (2014). Interactive Development of Cross-Device User Interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2793–2802. doi: [10.1145/2556288.2556980](https://doi.org/10.1145/2556288.2556980)

189. Nebeling, M., Nebeling, J., Yu, A., & Rumble, R. (2018). ProtoAR: Rapid Physical-Digital Prototyping of Mobile Augmented Reality Applications. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. doi: [10.1145/3173574.3173927](https://doi.org/10.1145/3173574.3173927)
190. Nebeling, M., & Norrie, M. (2012). jQMultiTouch: Light-weight Toolkit and Development Framework for Multi-Touch/Multi-Device Web Interfaces. *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 61–70. doi: [10.1145/2305484.2305497](https://doi.org/10.1145/2305484.2305497)
191. Nebeling, M., Teunissen, E., Husmann, M., & Norrie, M. C. (2014). XDKinect: Development Framework for Cross-Device Interaction Using Kinect. *Proceedings of the 2014 ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, 65–74. doi: [10.1145/2607023.2607024](https://doi.org/10.1145/2607023.2607024)
192. Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.
193. Nielsen, J., Fehr, I., & Nyman, H. O. (1991). The Learnability of Hypercard as an Object-Oriented Programming System. *Behaviour & Information Technology*, 10(2), 111–120. doi: [10.1080/01449299108924276](https://doi.org/10.1080/01449299108924276)
194. Nielsen, J., & Molich, R. (1990). Heuristic Evaluation of User Interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 249–256. doi: [10.1145/97243.97281](https://doi.org/10.1145/97243.97281)
195. Norman, D. A. (2013). *The Design of Everyday Things*. USA: Basic Books, Inc.

196. Olsen, D. R. (2007). Evaluating User Interface Systems Research. *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 251–258. doi: [10.1145/1294211.1294256](https://doi.org/10.1145/1294211.1294256)
197. Oulasvirta, A., & Hornbæk, K. (2016). HCI Research as Problem-Solving. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 4956–4967. doi: [10.1145/2858036.2858283](https://doi.org/10.1145/2858036.2858283)
198. Palmer, T. (2018). *Design Tools Survey*. Retrieved from <https://uxtools.co/survey-2018/>
199. Penner, R. (2002). *Robert Penner's Programming Macromedia Flash MX* (1st ed.). USA: McGraw-Hill, Inc.
200. Pfeiffer, M., Duente, T., & Rohs, M. (2016). Let Your Body Move: A Prototyping Toolkit for Wearable Force Feedback with Electrical Muscle Stimulation. *Proceedings of the 18th International Conference on Human-Computer Interaction with Mobile Devices and Services*, 418–427. doi: [10.1145/2935334.2935348](https://doi.org/10.1145/2935334.2935348)
201. Polson, P. G., Lewis, C., Rieman, J., & Wharton, C. (1992). Cognitive Walkthroughs: A Method for Theory-Based Evaluation of User Interfaces. *International Journal of Man-Machine Studies*, 36(5), 741–773. doi: [10.1016/0020-7373\(92\)90039-N](https://doi.org/10.1016/0020-7373(92)90039-N)
202. Pratt, A., & Nunes, J. (2012). *Interactive Design: An Introduction to the Theory and Application of User-centered Design*. Rockport Publishers.

203. Preece, Jennifer, Sharp, H., & Rogers, Y. (2015). *Interaction Design: Beyond Human-Computer Interaction*. John Wiley & Sons.
204. Preece, Jenny, & Rombach, H. D. (1994). A taxonomy for combining software engineering and human-computer interaction measurement approaches: Towards a common framework. *International Journal of Human-Computer Studies*, 41(4), 553–583. doi: [10.1006/ijhc.1994.1073](https://doi.org/10.1006/ijhc.1994.1073)
205. Pugh, S. (1991). *Total Design: Integrated Methods for Successful Product Engineering*. Addison-Wesley.
206. Ramakers, R., Anderson, F., Grossman, T., & Fitzmaurice, G. (2016). RetroFab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, 409–419. doi: [10.1145/2858036.2858485](https://doi.org/10.1145/2858036.2858485)
207. Ramakers, R., Todi, K., & Luyten, K. (2015). PaperPulse: An Integrated Approach for Embedding Electronics in Paper Designs. *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, 2457–2466. doi: [10.1145/2702123.2702487](https://doi.org/10.1145/2702123.2702487)
208. Raskin, J. (2000). *The Humane Interface: New Directions for Designing Interactive Systems*. Addison-Wesley Professional.
209. Reach, A. M., & North, C. (2017). The Signals and Systems Approach to Animation. *ArXiv:1703.00521 [Cs]*. Retrieved from <http://arxiv.org/abs/1703.00521>

210. Rittel, H. W., & Webber, M. M. (1973). Planning Problems are Wicked. *Polity*, 4(155), e169.
211. Rogers, Y. (2004). New Theoretical Approaches for HCI. *Annual Review of Information Science and Technology*, 38(1), 87–143.
212. Rogers, Y., & Marshall, P. (2017). Research in the Wild. *Synthesis Lectures on Human-Centered Informatics*, 10(3), i–97. doi: [10.2200/S00764ED1V01Y201703HCI037](https://doi.org/10.2200/S00764ED1V01Y201703HCI037)
213. Roseman, M., & Greenberg, S. (1996a). Building Real-Time Groupware with Groupkit, a Groupware Toolkit. *ACM Transactions on Computer-Human Interaction*, 3(1), 66–106. doi: [10.1145/226159.226162](https://doi.org/10.1145/226159.226162)
214. Roseman, M., & Greenberg, S. (1996b). TeamRooms: Groupware for Shared Electronic Spaces. *Conference Companion on Human Factors in Computing Systems*, 275–276. doi: [10.1145/257089.257319](https://doi.org/10.1145/257089.257319)
215. Rosenman, M. A., & Gero, J. S. (1989). Creativity in Design Using a Prototype Approach. *Design Computing Unit, Department of Architectural and Design Science, University of Sydney*, 1989. Pp. 207-232. CADLINE Has Abstract Only. Retrieved from <http://papers.cumincad.org/cgi-bin/works/aper/a442>
216. Saffer, D. (2013). *Microinteractions: Designing with Details*. O'Reilly Media, Inc.

217. Salber, D., Dey, A. K., & Abowd, G. D. (1999). The Context Toolkit: Aiding the Development of Context-Enabled Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 434–441. doi: [10.1145/302979.303126](https://doi.org/10.1145/302979.303126)
218. Salovaara, A., Oulasvirta, A., & Jacucci, G. (2017). Evaluation of Prototypes and the Problem of Possible Futures. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 2064–2077. doi: [10.1145/3025453.3025658](https://doi.org/10.1145/3025453.3025658)
219. Savage, V., Chang, C., & Hartmann, B. (2013). Sauron: Embedded Single-Camera Sensing of Printed Physical User Interfaces. *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, 447–456. doi: [10.1145/2501988.2501992](https://doi.org/10.1145/2501988.2501992)
220. Savage, V., Follmer, S., Li, J., & Hartmann, B. (2015). Makers' Marks: Physical Markup for Designing and Fabricating Functional Objects. *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, 103–108. doi: [10.1145/2807442.2807508](https://doi.org/10.1145/2807442.2807508)
221. Savage, V., Schmidt, R., Grossman, T., Fitzmaurice, G., & Hartmann, B. (2014). A Series of Tubes: Adding Interactivity to 3D Prints Using Internal Pipes. *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, 3–12. doi: [10.1145/2642918.2647374](https://doi.org/10.1145/2642918.2647374)

222. Savage, V., Zhang, X., & Hartmann, B. (2012). Midas: Fabricating Custom Capacitive Touch Sensors to Prototype Interactive Objects. *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, 579–588. doi: [10.1145/2380116.2380189](https://doi.org/10.1145/2380116.2380189)
223. Schilit, B., Adams, N., & Want, R. (1994). Context-Aware Computing Applications. *1994 First Workshop on Mobile Computing Systems and Applications*, 85–90. doi: [10.1109/WMCSA.1994.16](https://doi.org/10.1109/WMCSA.1994.16)
224. Schön, D. A. (1987). *The Reflective Practitioner*. Jossey-Bass San Francisco.
225. Sennett, R. (2008). *The Craftsman*. Penguin Books.
226. Seyed, T., Azazi, A., Chan, E., Wang, Y., & Maurer, F. (2015). SoD-Toolkit: A Toolkit for Interactively Prototyping and Developing Multi-Sensor, Multi-Device Environments. *Proceedings of the 2015 International Conference on Interactive Tabletops & Surfaces*, 171–180. doi: [10.1145/2817721.2817750](https://doi.org/10.1145/2817721.2817750)
227. Shen, C., Vernier, F. D., Forlines, C., & Ringel, M. (2004). DiamondSpin: An extensible toolkit for around-the-table interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 167–174. doi: [10.1145/985692.985714](https://doi.org/10.1145/985692.985714)
228. Shneiderman, B. (1983). Human Factors of Interactive Software. In A. Blaser & M. Zeppritz (Eds.), *Enduser Systems and Their Human Factors* (pp. 9–29). doi: [10.1007/3-540-12273-7_16](https://doi.org/10.1007/3-540-12273-7_16)

229. Smith, D. C. (1975). *PYGMALION: A Creative Programming Environment*. Retrieved from Stanford Univeristy website: <https://apps.dtic.mil/sti/citations/ADA016811>
230. Star, S. L., & Griesemer, J. R. (1989). Institutional Ecology, 'Translations' and Boundary Objects: Amateurs and Professionals in Berkeley's Museum of Vertebrate Zoology, 1907-39. *Social Studies of Science*, 19(3), 387-420. doi: [10.1177/030631289019003001](https://doi.org/10.1177/030631289019003001)
231. Stuerzlinger, W., Chapuis, O., Phillips, D., & Roussel, N. (2006). User Interface Façades: Towards Fully Adaptable User Interfaces. *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, 309-318. doi: [10.1145/1166253.1166301](https://doi.org/10.1145/1166253.1166301)
232. Subtraction.com. (2015). *The Tools Designers Are Using Today*. Retrieved from <https://tools.subtraction.com/>
233. Sutherland, I. E. (1964). Sketchpad a Man-Machine Graphical Communication System. *Simulation*, 2(5), R-3.
234. The Nature of Design Practice and Implications for Interaction Design Research. (n.d.). *International Journal of Design*. Retrieved from <http://ijdesign.org/index.php/IJDesign/article/view/240>
235. Tscheligi, M., Houde, S., Kolli, R., Marcus, A., Muller, M., & Mullet, K. (1995). Creative Prototyping Tools: What Interaction Designers Really Need to Produce Advanced User Interface Concepts. *Conference Companion on Human Factors in Computing Systems*, 170-171. doi: [10.1145/223355.223485](https://doi.org/10.1145/223355.223485)

236. van Oosterhout, A., Bruns Alonso, M., & Jumisko-Pyykkö, S. (2018). Ripple Thermostat: Affecting the Emotional Experience through Interactive Force Feedback and Shape Change. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1–12. doi: [10.1145/3173574.3174229](https://doi.org/10.1145/3173574.3174229)
237. Victor, B. (n.d.). *Inventing on Principle*. Retrieved from <http://worrydream.com/#!/InventingOnPrinciple>
238. Villar, N., & Gellersen, H. (2007). A Malleable Control Structure for Softwired User Interfaces. *Proceedings of the 1st International Conference on Tangible and Embedded Interaction*, 49–56. doi: [10.1145/1226969.1226980](https://doi.org/10.1145/1226969.1226980)
239. Wakita, A., & Anezaki, Y. (2010). Intuino: An Authoring Tool for Supporting the Prototyping of Organic Interfaces. *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, 179–188. doi: [10.1145/1858171.1858204](https://doi.org/10.1145/1858171.1858204)
240. Walny, J. (2016). *Thinking with Sketches: Leveraging Everyday Use of Visuals for Information Visualization* (Graduate Studies). Retrieved from <https://prism.ucalgary.ca/handle/11023/3499>
241. Walny, J., Lee, B., Johns, P., Henry Riche, N., & Carpendale, S. (2012). Understanding Pen and Touch Interaction for Data Exploration on Interactive Whiteboards. *IEEE Transactions on Visualization and Computer Graphics*, 18(12), 2779–2788. doi: [10.1109/TVCG.2012.275](https://doi.org/10.1109/TVCG.2012.275)

242. Wang, C., Yeh, H.-M., Wang, B., Wu, T.-Y., Tsai, H.-R., Liang, R.-H., ... Chen, M. Y. (2016). CircuitStack: Supporting Rapid Prototyping and Evolution of Electronic Circuits. *Proceedings of the 29th Annual ACM Symposium on User Interface Software and Technology*, 687–695. doi: [10.1145/2984511.2984527](https://doi.org/10.1145/2984511.2984527)
243. Wang, F., & Ren, X. (2009). Empirical evaluation for finger input properties in multi-touch interaction. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1063–1072. doi: [10.1145/1518701.1518864](https://doi.org/10.1145/1518701.1518864)
244. Wang, M., Boring, S., & Greenberg, S. (2012). Proxemic Peddler: A Public Advertising Display That Captures and Preserves the Attention of a Passerby. *Proceedings of the 2012 International Symposium on Pervasive Displays*. Presented at the New York, NY, USA. doi: [10.1145/2307798.2307801](https://doi.org/10.1145/2307798.2307801)
245. Weichel, C., Lau, M., & Gellersen, H. (2013). Enclosed: A Component-Centric Interface for Designing Prototype Enclosures. *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, 215–218. doi: [10.1145/2460625.2460659](https://doi.org/10.1145/2460625.2460659)
246. Weichel, C., Lau, M., Kim, D., Villar, N., & Gellersen, H. W. (2014). MixFab: A Mixed-Reality Environment for Personal Fabrication. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 3855–3864. doi: [10.1145/2556288.2557090](https://doi.org/10.1145/2556288.2557090)

247. Weiser, M. (1991). The Computer for the 21st Century. *Scientific American*, 265(3), 75–84.
248. Wiberg, M., & Stolterman, E. (2014). What Makes a Prototype Novel? A Knowledge Contribution Concern for Interaction Design Research. *Proceedings of the 8th Nordic Conference on Human-Computer Interaction: Fun, Fast, Foundational*, 531–540. doi: [10.1145/2639189.2639487](https://doi.org/10.1145/2639189.2639487)
249. Wiethoff, A., Schneider, H., Küfner, J., Rohs, M., Butz, A., & Greenberg, S. (2013). Paperbox: A Toolkit for Exploring Tangible Interaction on Interactive Surfaces. *Proceedings of the 9th ACM Conference on Creativity & Cognition*, 64–73. doi: [10.1145/2466627.2466635](https://doi.org/10.1145/2466627.2466635)
250. Wigdor, D., Benko, H., Pella, J., Lombardo, J., & Williams, S. (2011). Rock & Rails: Extending Multi-Touch Interactions with Shape Gestures to Enable Precise Spatial Manipulations. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1581–1590. doi: [10.1145/1978942.1979173](https://doi.org/10.1145/1978942.1979173)
251. Wimmer, R., Kranz, M., Boring, S., & Schmidt, A. (2007). A Capacitive Sensing Toolkit for Pervasive Activity Detection and Recognition. *Fifth Annual IEEE International Conference on Pervasive Computing and Communications (PerCom '07)*, 171–180. doi: [10.1109/PERCOM.2007.1](https://doi.org/10.1109/PERCOM.2007.1)

252. Wing, J. M. (n.d.). Computational thinking and thinking about computing. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*. doi: <https://doi.org/10.1098/rsta.2008.0118>
253. Wobbrock, J. O., & Kientz, J. A. (2016). Research Contributions in Human-Computer Interaction. *Interactions*, 23(3), 38–44. doi: [10.1145/2907069](https://doi.org/10.1145/2907069)
254. Wobbrock, J. O., Wilson, A. D., & Li, Y. (2007). Gestures Without Libraries, Toolkits or Training: A \$1 Recognizer for User Interface Prototypes. *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, 159–168. doi: [10.1145/1294211.1294238](https://doi.org/10.1145/1294211.1294238)
255. Wu, C.-J., Houben, S., & Marquardt, N. (2017). EagleSense: Tracking People and Devices in Interactive Spaces using Real-Time Top-View Depth-Sensing. *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, 3929–3942. doi: [10.1145/3025453.3025562](https://doi.org/10.1145/3025453.3025562)
256. Xiao, R., Harrison, C., & Hudson, S. E. (2013). WorldKit: Rapid and Easy Creation of Ad-Hoc Interactive Applications on Everyday Surfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 879–888. doi: [10.1145/2470654.2466113](https://doi.org/10.1145/2470654.2466113)
257. Yang, J., & Wigdor, D. (2014). Panelrama: Enabling Easy Specification of Cross-Device Web Applications. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 2783–2792. doi: [10.1145/2556288.2557199](https://doi.org/10.1145/2556288.2557199)

258. Yeh, T., Chang, T.-H., & Miller, R. C. (2009). Sikuli: Using GUI Screenshots for Search and Automation. *Proceedings of the 22nd Annual ACM Symposium on User Interface Software and Technology*, 183–192. doi: [10.1145/1622176.1622213](https://doi.org/10.1145/1622176.1622213)
259. Yu, N.-H., Tsai, S.-S., Hsiao, I.-C., Tsai, D.-J., Lee, M.-H., Chen, M. Y., & Hung, Y.-P. (2011). Clip-on Gadgets: Expanding Multi-Touch Interaction Area with Unpowered Tactile Controls. *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, 367–372. doi: [10.1145/2047196.2047243](https://doi.org/10.1145/2047196.2047243)
260. Zimmerman, J., Forlizzi, J., & Evenson, S. (2007). Research Through Design as a Method for Interaction Design Research in HCI. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 493–502. doi: [10.1145/1240624.1240704](https://doi.org/10.1145/1240624.1240704)