

GAINING DESIGN INSIGHT THROUGH INTERACTION PROTOTYPING TOOLS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF STANFORD UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Björn Hartmann
September 2009

© 2009 by Björn Hartmann
All Rights Reserved

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

(Scott R. Klemmer) Principal Adviser

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

(Terry Winograd)

I certify that I have read this dissertation and that in my opinion it is fully adequate, in scope and quality, as a dissertation for the degree of Doctor of Philosophy.

(Pat Hanrahan)

Approved for the University Committee on Graduate Studies:

ABSTRACT

Prototyping is the fundamental activity that structures innovation in design. While prototyping tools are now common for graphical user interfaces on personal computers, prototyping interactions for ubiquitous computing systems remains out of reach for designers. This dissertation contributes concepts and techniques, embodied in software and hardware artifacts, to answer two research questions:

- 1) How can design tools enable a wider range of designers to create functional prototypes of ubiquitous computing user interfaces?
- 2) How can design tools support the larger process of learning from these prototypes?

Fieldwork at professional design companies showed that design generalists lack the tools to fluently experiment with interactions for sensor-based interfaces and information appliances. The first contribution of this dissertation is a set of methods, embodied in authoring tools, that lower the expertise threshold required to author such novel interfaces. These tools enable more designers to author a wider range of interfaces, faster. Visual authoring of control flow diagrams and plug-and-play hardware linked to software abstractions for hardware components enable rapid authoring of interaction logic. This dissertation also introduces programming by demonstration techniques for sensor-based interactions to derive high-level events from continuous sensor data streams.

Enabling the construction of prototypes is an important function of design tools; however, it should not be the only goal. Prototypes are just a means to an end — they are built to elicit feedback about design choices. The second contribution of this thesis is a set of systems that explicitly support the design practices of exploration and iteration. Exploration is supported through enabling the creation of multiple, parallel user interface alternatives. The design-test-analysis loop of iterative design is supported through techniques for rapid review of user test data and techniques for revision of interaction diagrams. The presented work is informed by interviews and collaborations with professional interaction designers. The tools are evaluated through a combination of laboratory studies and deployments to interaction design students at Stanford and in industry.

ACKNOWLEDGMENTS

I would like to thank my advisor, Scott Klemmer, for the freedom and support he gave me to follow my interests and chart my own path over the last five years. Scott and I both arrived at Stanford in the Fall of 2004, and he took me on as his first newly-admitted student. I thank him for his trust, and the passion, commitment, and energy with which he led our research group. His door was always open and his advice for navigating graduate school and the research world prepared me well for my upcoming transition to faculty life. I also thank my other reading committee members, Terry Winograd and Pat Hanrahan, for their feedback and stimulating conversations over the years. I am grateful to Stu Card and John Haymaker, who served on my orals committee. Bill Verplank has been a source of inspiration throughout my time at Stanford. His course on building electronic musical instruments launched me in the direction of this dissertation.

Much of the work in this dissertation was undertaken with the help of a fantastic group of collaborators. I have had the privilege to supervise a group of talented and hard-working undergraduate summer interns through Stanford's CURIS program, who have contributed to every project presented here: Michael Bernstein, Loren Yu, Anthony Ricciardi, Timothy Cardenas, Sean Follmer, and Daniel MacDougall. Many other collaborators have worked with me on either the research presented in this dissertation, or the numerous other projects we pursued together. I am deeply indebted to them. At Stanford, I worked with Leith Abdulla, Abel Allison, Marcello Bastea-Forte, Joel Brandt, Jesse Cirimele, Kevin Collins, Scott Doorley, Wendy Ju, Michel Krieger, Dan Maynes-Aminzade, Nirav Mehta, Manas Mittal, Merrie Ringel Morris, Erica Robles, Leila Takayama, Leslie Wu, and Yeonsoo Yang. At Microsoft Research, Merrie Morris, Andy Wilson, and Hrvoje Benko were fabulous mentors.

My research would not have been possible without the tireless work of system administrator John Gerth and the lab administrative staff, Heather Gentner, Ada Glucksman, Melissa Rivera and Monica Niemic. I would also like to thank the professional designers, who shared their time and expertise with me; Arna Ionescu and Hans-Christoph Haenlein at IDEO were especially generous with their time.

Finally, I would like to express my gratitude to my parents, Volker and Lieselotte Hartmann, and my wife, Tania Treis. Your love and support has made this possible. Thank you.

I was supported by an Agilent School of Engineering Fellowship and a SAP Stanford Graduate Fellowship during my five years at Stanford. The d.tools project was further supported by a grant from the Stanford Office of Technology Licensing. Dai Nippon Printing, through the Stanford MediaX organization, supported the development of Exemplar. Juxtapose was partially funded through NSF grant IIS-0745320. Nokia and Intel donated hardware for multiple projects.

TABLE OF CONTENTS

| | | |
|-----------|----------------------------------------------------------------------|----|
| CHAPTER 1 | INTRODUCTION | 1 |
| 1.1 | Thesis Contributions | 2 |
| 1.2 | Dissertation Roadmap | 4 |
| 1.2.1 | Background: Prototypes in the Design Process (Chapter 2) | 4 |
| 1.2.2 | Related Work (Chapter 3) | 4 |
| 1.2.3 | Authoring Sensor-Based Interactions (Chapter 4) | 5 |
| 1.2.4 | Creating Alternative Design Solutions (Chapter 5) | 7 |
| 1.2.5 | Gaining Insight Through Feedback (Chapter 6) | 9 |
| 1.2.6 | Conclusions & Future Work (Chapter 7) | 10 |
| 1.2.7 | Overview: Research Concerns & Projects | 10 |
| 1.3 | Statement on Multiple Authorship and Prior Publications | 11 |
| CHAPTER 2 | BACKGROUND: PROTOTYPES IN THE DESIGN PROCESS | 12 |
| 2.1 | Design, Defined | 12 |
| 2.1.1 | What Do We Mean By Design? | 12 |
| 2.1.2 | A Short History of Professional Design | 14 |
| 2.1.3 | How Do Designers Work? Models of the Design Process | 14 |
| 2.2 | Understanding Prototypes | 15 |
| 2.2.1 | Prototypes, Defined | 16 |
| 2.2.2 | Benefits of Prototyping | 16 |
| 2.2.2.1 | Quantifying the Value of Prototyping | 17 |
| 2.2.2.2 | Cognitive Benefits of Prototyping | 17 |
| 2.2.2.3 | Reflective Practice: The Value of Surprise | 18 |
| 2.2.2.4 | Prototyping as a Teaching Technique | 19 |
| 2.2.3 | The Purpose of Prototyping — Design Perspectives | 19 |
| 2.2.3.1 | What Do Prototypes Prototype? | 19 |
| 2.2.3.2 | Experience Prototyping | 20 |
| 2.2.3.3 | Inspiration, Evolution, Validation | 21 |
| 2.2.3.4 | Prototyping as Inquiry | 22 |
| 2.2.3.5 | Low-Fidelity Prototypes Might Be Preferable | 22 |
| 2.2.4 | The Purpose of Prototyping — Software Engineering Perspectives | 23 |
| 2.2.4.1 | Exploration, Experimentation, Evolution | 24 |
| 2.2.4.2 | Prototypes as Immature Products | 25 |

| | | |
|------------------------------------|-----------------------------------------------------------------------|-----------|
| 2.2.4.3 | Presentation Prototypes, Breadboards, and Pilot Systems..... | 25 |
| 2.2.4.4 | Capturing and Sharing Knowledge Gained from Prototypes..... | 26 |
| 2.2.5 | Synthesis of the Surveyed Material..... | 26 |
| CHAPTER 3 RELATED WORK..... | | 29 |
| 3.1 | Status Quo: Tools & Industry Practices Today..... | 29 |
| 3.1.1 | Building Prototypes..... | 29 |
| 3.1.1.1 | Desktop-Based User Interfaces..... | 29 |
| 3.1.1.2 | Non-Traditional User Interfaces..... | 31 |
| 3.1.2 | Gaining Insight from Prototypes..... | 33 |
| 3.1.2.1 | Considering Alternatives..... | 33 |
| 3.1.2.2 | Annotating and Reviewing..... | 34 |
| 3.1.2.3 | Feedback from User Tests..... | 34 |
| 3.2 | UI Prototyping Tools..... | 35 |
| 3.3 | Tool Support for Physical Computing..... | 41 |
| 3.4 | Visual Authoring..... | 46 |
| 3.4.1 | Visual Formalisms..... | 46 |
| 3.4.1.1 | State Diagrams..... | 47 |
| 3.4.1.2 | Statecharts..... | 47 |
| 3.4.1.3 | Flowcharts..... | 48 |
| 3.4.1.4 | Data Flow Diagrams..... | 49 |
| 3.4.1.5 | Unified Modeling Language..... | 49 |
| 3.4.2 | Visual Programming Proper..... | 50 |
| 3.4.2.1 | Control Flow Languages..... | 50 |
| 3.4.2.2 | Data Flow Languages..... | 52 |
| 3.4.2.3 | Control Flow and Data Flow in d.tools and Exemplar..... | 53 |
| 3.4.3 | Enhanced Editing Environments..... | 54 |
| 3.4.3.1 | Visual Editors..... | 54 |
| 3.4.3.2 | Structured Source Editors..... | 55 |
| 3.4.3.3 | Hybrid Environments..... | 56 |
| 3.4.4 | Analyzing Visual Languages with Cognitive Dimensions of Notation..... | 56 |
| 3.5 | Programming by Demonstration..... | 58 |
| 3.5.1 | PBD on the Desktop..... | 58 |
| 3.5.2 | PBD for Ubiquitous Computing..... | 58 |
| 3.6 | Designing Multiple Alternatives & Rapid Exploration..... | 60 |
| 3.6.1 | Tools for Working with Alternatives in Parallel..... | 60 |
| 3.6.2 | Rapid Sequential Modification..... | 62 |

| | | |
|--------------------------------------------------------------|------------------------------------------------------------------------|-----------|
| 3.7 | Feedback from User Testing..... | 63 |
| 3.7.1 | Improving Work with Usability Videos..... | 64 |
| 3.7.2 | Integrating Design, Test & Analysis..... | 65 |
| 3.8 | Team Feedback & UI Revision..... | 65 |
| 3.8.1 | Annotation Tools..... | 66 |
| 3.8.2 | Difference Visualization Tools | 66 |
| 3.8.3 | Capturing Design History..... | 67 |
| CHAPTER 4 AUTHORIZING SENSOR-BASED INTERACTIONS | | 68 |
| 4.1 | Authoring Physical User Interfaces with d.tools..... | 68 |
| 4.1.1 | Fieldwork | 69 |
| 4.1.2 | Design Principles | 70 |
| 4.1.3 | Prototyping with d.tools | 71 |
| 4.1.3.1 | Designing Physical Interactions with ‘Plug and Draw’ | 72 |
| 4.1.3.2 | Authoring Interaction Models | 73 |
| 4.1.3.3 | Raising the Complexity Ceiling of Prototypes..... | 74 |
| 4.1.4 | Architecture and Implementation | 76 |
| 4.1.4.1 | Plug-and-Play Hardware..... | 76 |
| 4.1.4.2 | Hardware Extensibility | 77 |
| 4.1.4.3 | Software | 79 |
| 4.1.5 | Evaluation..... | 82 |
| 4.1.5.1 | Establishing Threshold with a First Use Study | 82 |
| 4.1.5.2 | Rebuilt Existing and Novel Devices | 85 |
| 4.1.5.3 | Teaching Experiences — HCI Design Studio | 87 |
| 4.1.6 | d.tools Mobile..... | 89 |
| 4.1.7 | Limitations & Extensions..... | 92 |
| 4.1.7.1 | Dynamic Graphics Require Scripting | 92 |
| 4.1.7.2 | Hierarchical Diagrams Not Supported | 93 |
| 4.1.7.3 | Screen Real Estate Not Used Efficiently | 93 |
| 4.1.7.4 | Lack of Support for Actuation | 94 |
| 4.1.7.5 | Prototypes Have to be Tethered to PC by Wire | 94 |
| 4.2 | Exemplar: Programming Sensor-Based Interactions by Demonstration | 96 |
| 4.2.1 | Sensor-Based Interactions | 98 |
| 4.2.1.1 | Binary, Categorical, and Continuous Signals..... | 98 |
| 4.2.1.2 | Working with Continuous Signals | 98 |
| 4.2.1.3 | Generating Discrete Events | 99 |
| 4.2.2 | Design Principles..... | 99 |

| | | |
|--------------------------------------------------------------|----------------------------------------------------------------|------------|
| 4.2.3 | Designing with Exemplar | 101 |
| 4.2.3.1 | Peripheral Awareness | 102 |
| 4.2.3.2 | Drilling Down and Filtering | 102 |
| 4.2.3.3 | Demonstration and Mark-Up | 103 |
| 4.2.3.4 | Recognition and Generalization..... | 103 |
| 4.2.3.5 | Event Output | 105 |
| 4.2.3.6 | Many Sensors, Many Events..... | 105 |
| 4.2.3.7 | Demonstrate-Edit-Review | 106 |
| 4.2.4 | Implementation & Architecture | 106 |
| 4.2.4.1 | Signal Input, Output, and Display | 106 |
| 4.2.4.2 | Pattern Recognition..... | 107 |
| 4.2.4.3 | Extensibility..... | 107 |
| 4.2.5 | Evaluation..... | 108 |
| 4.2.5.1 | Cognitive Dimensions Usability Inspection | 108 |
| 4.2.5.2 | First-Use Study..... | 111 |
| 4.2.5.3 | Using Exemplar to Create Game Controllers..... | 115 |
| 4.2.6 | Limitations & Extensions..... | 116 |
| 4.2.6.1 | Lack of Support for Other Time Series Data | 116 |
| 4.2.6.2 | Matching Performance Degrades for Multi-Dimensional Data | 117 |
| 4.2.6.3 | Lack of Visualization Support for Multi-Dimensional Data | 117 |
| 4.2.6.4 | Lack of Support for Parameter Estimation | 118 |
| 4.2.6.5 | Difficult to Interpret Sensor Data History..... | 118 |
| CHAPTER 5 CREATING ALTERNATIVE DESIGN SOLUTIONS | | 120 |
| 5.1 | Alternatives in Juxtapose | 120 |
| 5.2 | Formative Interviews..... | 122 |
| 5.3 | Exploring Options with Juxtapose..... | 123 |
| 5.4 | Architecture for Alternative Design | 124 |
| 5.4.1 | Parallel Editing..... | 125 |
| 5.4.2 | Parallel Execution and Tuning | 126 |
| 5.4.3 | Writing Tunable Code..... | 131 |
| 5.4.3.1 | Hardware Support | 131 |
| 5.5 | User Experiences with Juxtapose | 132 |
| 5.5.1 | Method..... | 133 |
| 5.5.2 | Results..... | 134 |
| 5.6 | Limitations & Extensions | 136 |
| 5.6.1 | Will Designers Really Benefit from Linked Sources? | 137 |

| | | |
|---------------------------------------------------------|-----------------------------------------------------------------------|------------|
| 5.6.2 | Is Tuning of Numbers and Booleans Sufficient? | 138 |
| 5.6.3 | Are Code Alternatives Enough? | 138 |
| 5.6.4 | Alternatives for Complex Code Bases | 139 |
| 5.6.5 | Support Exploration at the Language Level | 139 |
| 5.6.6 | Integrate With Testing..... | 140 |
| 5.7 | Supporting Alternatives in Visual Programs..... | 140 |
| CHAPTER 6 GAINING INSIGHT THROUGH FEEDBACK | | 143 |
| 6.1 | Feedback in User Testing: Supporting Desing-Test-Analyze Cycles | 143 |
| 6.1.1 | Testing Prototypes..... | 144 |
| 6.1.2 | Analyzing Test Sessions | 146 |
| 6.1.2.1 | Single User Analysis | 146 |
| 6.1.2.2 | Group Analysis..... | 148 |
| 6.1.3 | Implementation | 149 |
| 6.1.4 | Limitations & Extensions..... | 149 |
| 6.1.4.1 | No Support for Quantitative Analysis | 149 |
| 6.1.4.2 | Limited Visibility of Application Behavior During Test | 150 |
| 6.1.4.3 | Cannot Compare Multiple Prototypes in Analysis Mode..... | 150 |
| 6.1.4.4 | Limited Query Language | 151 |
| 6.1.4.5 | Interaction Techniques Have Not Been Formally Evaluated..... | 151 |
| 6.2 | Capturing Feedback from Other Designers: d.note | 152 |
| 6.2.1 | Revision Practices In Other Domains..... | 153 |
| 6.2.2 | A Visual Language for Revising Interactions..... | 155 |
| 6.2.2.1 | Revising Behavior | 155 |
| 6.2.2.2 | Revising Appearance | 157 |
| 6.2.2.3 | Revising Device Definition..... | 157 |
| 6.2.2.4 | Commenting | 158 |
| 6.2.2.5 | Proposing Alternatives..... | 158 |
| 6.2.3 | Scenario..... | 158 |
| 6.2.4 | The d.note Java Implementation..... | 159 |
| 6.2.4.1 | Specifying Actions Through Stylus Input..... | 159 |
| 6.2.5 | Evaluation: Comparing Interactive & Static Revisions..... | 160 |
| 6.2.5.1 | Study 1: Authoring Revisions | 160 |
| 6.2.5.2 | Study 2: Interpreting Revisions | 165 |
| 6.2.6 | Limitations & Extensions..... | 167 |
| 6.2.6.1 | Cannot Comment on Dynamic Behavior..... | 168 |
| 6.2.6.2 | Cannot Revise Dynamic Behavior | 168 |

| | | |
|---------------------------------------------------|---------------------------------------------------------------------|------------|
| 6.2.6.3 | How To Support Identified Revision Principles for Source Code?..... | 169 |
| CHAPTER 7 CONCLUSIONS AND FUTURE WORK..... | | 170 |
| 7.1 | Restatement of Contributions | 170 |
| 7.2 | Future Work..... | 171 |
| 7.2.1 | Design Tools That Support Collaboration | 172 |
| 7.2.2 | Authoring by Example Modification | 173 |
| 7.2.2.1 | Finding Examples..... | 174 |
| 7.2.2.2 | Synthesizing Examples | 174 |
| 7.2.2.3 | Extracting Examples | 175 |
| 7.2.2.4 | Integrating Examples | 175 |
| 7.2.3 | Authoring Off the Desktop | 176 |
| 7.2.3.1 | Going Large: New Studio Spaces for Interaction Design | 176 |
| 7.2.3.2 | Going Small: Authoring on Handheld Devices | 178 |
| 7.2.4 | Designing Device Ecologies | 179 |
| 7.3 | Closing Remarks..... | 180 |
| REFERENCES | | 181 |

LIST OF FIGURES

| | | |
|-------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 1.1: | The d.tools visual authoring environment enables rapid construction of UI logic..... | 6 |
| Figure 1.2: | The d.tools hardware interface offers a plug-and-play architecture for interface components..... | 6 |
| Figure 1.3: | Exemplar combines programming-by-demonstration with direct manipulation to author sensor-based interactions..... | 7 |
| Figure 1.4: | This evaluation participant used Exemplar to control 2D aiming in a game with an accelerometer, and shooting with the flick of a bend sensor..... | 7 |
| Figure 1.5: | Side-by-side execution in Juxtapose enables rapid comparison of alternatives..... | 8 |
| Figure 1.6: | Juxtapose automatically generates control interfaces for program variables..... | 8 |
| Figure 1.7: | d.note introduces stylus-driven revision of interaction diagrams..... | 9 |
| Figure 1.8: | The d.tools test & analysis functions link video clips of test sessions to event traces of the tested prototype..... | 9 |
| Figure 2.1: | Design process stages according to Moggridge [189]. Diagram redrawn by the author..... | 15 |
| Figure 2.2: | The Houde & Hill model distinguishes <i>Role</i> , <i>Implementation</i> , and <i>Look and Feel</i> functions of prototypes. (Diagram redrawn by the author)..... | 20 |
| Figure 2.3: | The IDEO three-stage model of prototyping: as a design project progresses, the number of entertained ideas decreases, and prototypes turn from inspiration tools to validation tools. Diagram redrawn by the author..... | 21 |
| Figure 2.4: | Why are prototypes constructed in design?..... | 26 |
| Figure 2.5: | What aspects of a product can prototypes approximate?..... | 27 |
| Figure 2.6: | What kind of functionality can prototypes exhibit?..... | 27 |
| Figure 3.1: | Common tools used for UI prototyping as reported in Myers' survey of interaction designers [195]. Figure redrawn by the author..... | 30 |
| Figure 3.2: | Pering's "Buck" for testing PDA applications: PDA hardware is connected to a laptop using a custom hardware interface. Application output is shown on the laptop screen..... | 32 |
| Figure 3.3: | IDEO interaction prototype for a digital camera UI. The handheld prototype is driven by the desktop computer in the background..... | 32 |

| | | |
|--------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----|
| Figure 3.4: | Buxton’s Doormouse [56] is an example of a “hardware hack” that repurposes a standard mouse. | 33 |
| Figure 3.5: | Timeline of prototyping tools for graphical user interfaces. | 37 |
| Figure 3.6: | Bailey’s DEMAIS system introduced a visual language for sketching multimedia applications [40]..... | 38 |
| Figure 3.7: | Li’s Topiary system for prototyping location-aware mobile applications [163]..... | 38 |
| Figure 3.8: | Timeline of selected physical computing toolkits..... | 42 |
| Figure 3.9: | A partial, hierarchical statechart for a wrist watch with alarm function; redrawn from an example in Harel [105]. | 47 |
| Figure 3.10: | Example of a flowchart, adapted from Glinert [87]. | 48 |
| Figure 3.11: | Example of a Nassi-Shneiderman structogram, adapted from Glinert [87]. | 48 |
| Figure 3.12: | Example of a data flow diagram, redrawn by the author from Yourdon [259: p. 159]..... | 49 |
| Figure 3.13: | Examples of commercial or open source data flow languages. A: Quartz Composer; B: Pure Data; C: Yahoo Pipes | 52 |
| Figure 3.14: | Example of hybrid authoring in Pure Data: a visual node contains an algebraic expression. | 56 |
| Figure 3.15: | SUEDE introduced techniques to unite design, test, and analysis of speech user interfaces. | 65 |
| Figure 4.1: | Overview of prototyping with d.tools: A designer interacts both with a hardware prototype (left) and the authoring environment (right)..... | 68 |
| Figure 4.2: | The d.tools authoring environment. A: device designer. B: storyboard editor. C: GUI editor. D: asset library. E: property sheet..... | 71 |
| Figure 4.3: | d.tools plug-and-play: inserting a physical component causes a corresponding virtual component to appear in the d.tools device designer. | 72 |
| Figure 4.4: | d.tools interaction techniques. A: creating new transitions through dragging. B: adding a new condition to an existing transition. C: Visualizing sensor signal input and thresholds in context. D: parallel active states. E: editing code attached to a state..... | 74 |
| Figure 4.5: | The d.tools hardware interface (left). Individual smart components (middle) are can be plugged into any bus connector (right)..... | 76 |
| Figure 4.6: | Schematic diagram of the d.tools hardware infrastructure. Smart components are networked on an I2C bus. A master microcontroller | |

| | | |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| | communicates over a serial-over-USB connection with the computer running the d.tools authoring environment. | 77 |
| Figure 4.7: | Code examples for the d.tools scripting API. | 80 |
| Figure 4.8: | Task completion times, and prior experience and expertise of d.tools study participants. Participants completed task 1 in an average of 9 minutes, and task 2 in an average of 24 minutes. These times demonstrate that prototyping with d.tools is fast enough to be appropriate for early-stage design. | 83 |
| Figure 4.9: | Post-test survey results from the d.tools user study. Participants provided responses on Likert scales. | 84 |
| Figure 4.10: | Some applications built with d.tools in our research group. A: digital camera image navigation. B: sensor-enhanced smart PDA. C & D: tangible drawers for a multi-user interactive tabletop. E: proxemics-aware whiteboard. F: TiltType for orientation-based text entry. | 86 |
| Figure 4.11: | Some student projects built with d.tools. A: a tangible color mixing device where virtual color can be poured from physical paint buckets by tilting them over an LCD screen. B: a message recording system for children to exchange secrets. C: a smart clothes rack can detect which hangers are removed from the rack and display fashion advice on a nearby screen. D: a mobile shopping assistant can scan barcodes of grocery items and present sustainability information relating to the scanned item on its screen. E: a tangible audio mixer to produce cell phone ring tones. F: an accelerometer- equipped golf club used as a game controller. | 88 |
| Figure 4.12: | A d.tools mobile prototype on a Nokia N93 smart phone, with the storyboard logic of the prototype in the background. | 89 |
| Figure 4.13: | The d.tools mobile system architecture uses socket communication over a wireless connection to receive input events and send output commands to a smart phone. | 90 |
| Figure 4.14: | Iterative programming by demonstration for sensor-based interactions: A designer performs an action; annotates its recorded signal in Exemplar; tests the generated behavior; and exports it to d.tools. | 97 |
| Figure 4.15: | The Exemplar authoring environment offers visualization of live sensor data and direct manipulation techniques to interact with that data. | 101 |
| Figure 4.16: | Sensor data flows from left to right in the Exemplar UI. | 101 |

| | | |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Figure 4.17: | Exemplar shows output of the pattern matching algorithm on top of the sensor signal (in orange). When the graph falls below the threshold line, a match event is fired. | 104 |
| Figure 4.18: | Exemplar study setup: participants were seated at a dual monitor workstation in front of a large wall display. | 111 |
| Figure 4.19: | Self-reported prior experience of Exemplar study participants. | 111 |
| Figure 4.20: | Exemplar post-experiment questionnaire results. Error bars indicate ½ standard deviation in each direction. | 113 |
| Figure 4.21: | Interaction designs from the Exemplar user study. A: turning on blinkers by detecting head tilt with bend sensors; B: accelerometer used as continuous 2D head mouse; C: aiming and shooting with accelerometer and bend sensor; D: navigation through full body movement; E: bi-pedal navigation through force sensitive resistors; F: navigation by hitting the walls of a booth. | 113 |
| Figure 4.22: | Example of one study participant’s exploration: the participant created two different navigation schemes and two iterations on a trigger control; he tested his design on a target game three times within 16 minutes. | 114 |
| Figure 4.23: | Exemplar was used for public gaming installations at the San Mateo Maker Faire and at CHI 2007. For the CHI installation, wireless accelerometers were disguised as plush characters; the characters could be attached to clothing or objects in the environment. Characters and game concept were developed by Haiyan Zhang. | 116 |
| Figure 4.24: | A possible visualization for 2D thresholding in Exemplar. | 118 |
| Figure 5.1: | Design alternates between divergent and convergent stages. Diagram due to Buxton [55], redrawn by the author. | 120 |
| Figure 5.2: | Interaction designers explore options in Juxtapose through a source code editor that supports alternative code documents (left), a runtime interface that offers parallel execution and tuning of application parameters (center), and an external controller for spatially multiplexed input (right). | 121 |
| Figure 5.3: | In the Juxtapose source editor (left), users work with code alternatives in tabs. Users control whether modifications affect all alternatives or just the presently active alternative through linked editing. In the runtime interface (right), alternatives are executed in parallel. Designers tune application parameters with automatically generated control widgets. | 121 |

| | | |
|--------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Figure 5.4: | Example code from our inquiry: two behaviors co-exist in the same function body. The participant would switch between alternatives by changing which lines were commented. | 123 |
| Figure 5.5: | UI vignettes for the Juxtapose Scenario..... | 124 |
| Figure 5.6: | Juxtapose’s implementation of linked editing is based on maintaining block correspondences between alternatives across document modifications. | 125 |
| Figure 5.7: | Runtime tuning is achieved through bi-directional communication between a library added to the user’s application and the Juxtapose runtime user interface. | 127 |
| Figure 5.8: | When using Juxtapose mobile, code alternatives are executed on different phones in parallel. Variable tuning is accomplished through wireless communication. | 128 |
| Figure 5.9: | Two prototypes built with Juxtapose mobile. Left: A map navigation application explored use of variable tuning. Right: Two alternatives of a fisheye menu navigation technique running on two separate phones..... | 128 |
| Figure 5.10: | For microcontroller applications, Juxtapose transparently swaps out binary alternatives using a bootloader. Tuning is accomplished through code wrapping..... | 129 |
| Figure 5.11: | The pre-compilation processing step extracts variable declarations and emits them back into source code as a symbol table. | 130 |
| Figure 5.12: | Example application demonstrating live tuning of color parameters of a smart multicolor LED through the Juxtapose runtime user interface..... | 130 |
| Figure 5.13: | An external controller enables rapid surveying of multidimensional spaces. Variables names are projected on top of assigned controls to facilitate mapping. | 132 |
| Figure 5.14: | Study participants were given a code example that generates images of trees. They were asked to then match the four tree images shown above. | 133 |
| Figure 5.15: | Study participants were faster in completing the tree matching task with Juxtapose than without..... | 134 |
| Figure 5.16: | Study participants performed many more design parameter changes per minute with Juxtapose than without. | 134 |
| Figure 5.17: | A design space for exploring program alternatives. Choices implemented by Juxtapose are shown with a shaded background. | 137 |

| | | |
|--------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Figure 5.18: | Schematic of state alternatives in d.tools: alternatives are encapsulated in a common container. One alternative is active at a time. Alternatives have different output and different outgoing transitions. | 141 |
| Figure 5.19: | Screenshot of a d.tools container with two state alternatives. In the right alternative, screen graphics have been revised. | 141 |
| Figure 6.1: | d.tools supports design, test & analysis stages through integration with a video editor. | 143 |
| Figure 6.2: | Testing a prototype built with d.tools: A camera (A) is aimed at the tester and the physical prototype (B), which is driven by a storyboard (C) in d.tools. Live video of the test is recorded in the video editor (D) and annotated with events and state changes (E). Designers can add additional events to the record with a control console (F). | 144 |
| Figure 6.3: | The video recording interface in test mode. A: Active states at any point in time are encoded in a timeline view. B: Discrete input events show up as instantaneous events or press/release pairs. C: Continuous input data is visualized in-situ as a small graph in the timeline. | 145 |
| Figure 6.4: | In analysis mode, a dual-screen workstation enables simultaneous view of state model and video editor. | 146 |
| Figure 6.5: | Line thickness in analysis mode shows how many times a given transition was taken. | 147 |
| Figure 6.6: | Two query techniques link storyboard and video. A: Selecting a video segment highlights the state that was active at that time. B: Selecting a state in analyze mode highlights the corresponding video segment(s). | 147 |
| Figure 6.7: | Designers can <i>query by demonstration</i> : Generating input events in analyze mode filters recorded video so that only those sections where similar events were received are shown. | 147 |
| Figure 6.8: | Group analysis mode aggregates video and event data of multiple user sessions into one view. | 148 |
| Figure 6.9: | d.note enables interaction designers to revise and test functional prototypes of information appliances using a stylus-driven interface to d.tools. | 152 |
| Figure 6.10: | Interlinear revision tracking and comment visualization in word processing. | 153 |
| Figure 6.11: | Source code comparison tools show two versions of a file side-by-side. | 154 |
| Figure 6.12: | Video game designers draw annotations directly on rendered still images (from [55:p. 179]). | 154 |

| | | |
|--------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Figure 6.13: | States added during revision are rendered in blue. | 156 |
| Figure 6.14: | New screen graphics can be sketched in states. | 156 |
| Figure 6.15: | State deletions are rendered in red. Connections are marked as inactive..... | 156 |
| Figure 6.16: | Transition deletions are marked with a red cross and dashed red lines. | 156 |
| Figure 6.17: | Comments can be attached to any state. | 156 |
| Figure 6.18: | Alternative containers express different options for a state..... | 156 |
| Figure 6.19: | Sketched updates to screen content are immediately visible on attached hardware..... | 157 |
| Figure 6.20: | Changes to the device configuration are propagated to all states. Here, one button was deleted while two others were sketched in..... | 157 |
| Figure 6.21: | The d.note gesture set for stylus operation. Any stroke not interpreted as one of the first four actions is treated as a comment. | 159 |
| Figure 6.22: | Participants were given a prototype device with a color display and button input. They were asked to revise designs for a keychain display and a digital camera, both running on the provided device. | 161 |
| Figure 6.23: | Participants in study 1 revised d.tools designs on a large tablet display..... | 161 |
| Figure 6.24: | Two pairs of revision diagrams produced by our study participants. Diagrams produced with Sketchbook Pro in the control condition are shown on the left; diagrams produced with d.note are shown on the right. | 163 |
| Figure 6.25: | A design space of user interface revision tools. The sub-space d.note explored is highlighted in green. | 168 |
| Figure 7.1: | HelpMeOut offers asynchronous collaboration to suggest corrections to programming errors. 1: IDE instrumentation extracts bug fixes from programming sessions to a remote database. 2: Other programmers query the database when they encounter errors. 3: Suggested fixes are shown inside their IDE. | 175 |
| Figure 7.2: | The Pictionary table supports co-located design team work through multi-touch, multi-device input and overhead image capture. | 177 |

LIST OF TABLES

| | | |
|------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----|
| Table 1.1: | An overview how research concerns map onto the concrete systems presented in this dissertation..... | 10 |
| Table 2.1: | Three purposes of prototypes according to Floyd [79] (table redrawn from Schneider's summary [220])...... | 24 |
| Table 3.1: | Comparison of prior research in UI prototyping tools. | 36 |
| Table 3.2: | Comparison of prior research in physical computing tools. | 42 |
| Table 3.3: | The main dimensions of the Cognitive Dimensions of Notation inspection method (from [90: p.11]). | 57 |
| Table 3.4: | Differences between Design Galleries, set-based interaction, and Juxtapose are based on requirements of real-time input, method of alternative generation, and the source of input-output mapping. | 61 |
| Table 4.1: | The d.tools Java API allows designers to extend visual states with source code. The listed functions serve as the interface between designers' code and the d.tools runtime system. Standard Java classes are also accessible..... | 79 |
| Table 4.2: | The d.tools scripting API provides both global and object-oriented functions to interact with hardware, and a concise object-oriented set of function for manipulating GUI elements..... | 81 |
| Table 4.3: | Comparison of d.tools mobile and related mobile prototyping tools. | 92 |
| Table 6.1: | Content analysis of d.tools diagrams reveals different revision patterns: with d.note, participants wrote less and deleted more..... | 162 |
| Table 6.2: | Most frequently mentioned advantages and disadvantages of using d.note to <i>express</i> revisions. | 162 |
| Table 6.3: | How well could study 2 participants interpret the revisions created by others? Each vertical bar is one instance..... | 166 |
| Table 6.4: | Perceived advantages and disadvantages of using d.note to <i>interpret</i> revisions as reported by study participants. | 166 |