2018 | Faculty of Sciences

UHASSELT
KNOWLEDGE IN ACTION

Maastricht University

Doctoral dissertation submitted to obtain the degree of
Doctor of Science: Information Technology, to be defended by

Kashyap Todi

DOCTORAL DISSERTATION

Improving and Facilitating the Placement of Interactive Elements on User Interfaces

UHASSELT EDM

D/2018/2451/41

Promoter: Prof. Dr Kris Luyten | UHasselt

Co-promoter: Prof. Dr Andrew Vande Moere | KU Leuven

Acknowledgements

If you told my school teachers, "*Kashyap has received the degree of "Doctor of Science"*, some (or several) of them would be a tad bit shocked. However, I managed to pave my path from being a notorious school kid to a *serious* researcher over the course of years. This would have undeniably not been possible as a solo-mission: several people have supported, guided, and given me company along the way. I'd like to thank some of them here, and acknowledge their contribution.

Firstly, I want to thank Prof. Dr. Kris Luyten, my promoter, for his guidance and support over the entire course of my doctoral research. Kris has been a great mentor and advisor to me. He gave me enough room to explore different ideas, and did not restrict me to one path. Kris always had great advice and suggestions for tackling different research problems—be it for adding flesh to a topic or coming up with fun titles for a project. He also made sure to be available and prompt in responding with feedback: there have been so many occasions where I sent him an e-mail in the middle of the night, expecting a response in a day or two, and yet, found a reply in my inbox at around 2 A.M.

Second, I want to thank Prof. Dr. Antti Oulasvirta, who is one of the members of my jury, and also advised me during two sum-

mer internships over the course of my research. These internships, at the User Interface Group, were very enriching experiences—Antti is a great mentor and collaborator, and I have learnt a lot from him over the years. The rest of group members also contributed in making my stays pleasant and insightful. I look forward to continuing my path, in the immediate future, as a postdoctoral researcher, in his group.

Next, I want to thank Prof. Dr. Andrew Vande Moere, my co-promoter, for his involvement and advise during my PhD research. Andrew provided me with good advise, and suggestions, at crucial points. I visited his group in KU Leuven a few times during the early parts of my PhD, and we had some fruitful discussions. He also provided me with critical reflections on my thesis work when needed, and gave me several useful comments over the course of years. During the final stretch, his comments on my thesis drafts made me think deeper about my work, and helped the dissertation reach its final state. Moving on, I would like to thank all the other members of my jury: Prof. Dr. Karin Coninx, Prof. Dr. Jean Vanderdonckt, Dr. Emmanuel Pietriga, and Prof. Dr. Frank Van Reeth: Thank you all for your valuable time and constructive feedback in the final stages, which have helped me to improve this dissertation. I would also like to thank the chairman of my jury, Prof. Dr. Marc Gyssens.

This thesis would not have been possible without some amazing collaborators and colleagues, and I would like to thank them as well. I would like to especially mention Prof. Dr. Raf Ramakers, who I also shared an office with, for being a great collaborator and leading the PaperPulse project with such enthusiasm. I collaborated with Dr. Daryl Weir and Dr. Jussi Jokinen during my two internships at the User In-

terfaces group, and would like to thank them as well. I also want to thank Prof. Dr. Johannes Schöning, who worked with me on Bin-Put; Johannes was always keen on exploring new ideas, and added energy to our conversations. Several other people assisted me during my research, and I would like to thank all of them. All my colleagues at EDM have made it a pleasant and inviting place to work in, rich with interactions—be it lunch-time conversations, coffee-machine encounters, or occasional bar visits. I would like to explicitly mention Jens Bruhlmans for the excessive conversations about music or cryptocurrencies; Dr. Gustavo Rovelo for things related to Mexican food (and Tequila); my other officemates—Dr. Florian Heller, Dr. Davy Vanacken, and Dr. Pavel Samsonav; and Ingrid Konings, Hilde Wijnen, and Roger Claes for their assistance with administrative issues. I also thank all other colleagues, who I have not mentioned by name here, but have contributed in their own ways.

I would also like to thank my parents and family for supporting and encouraging me all the way, and in every move I have made. Without them, I would have never managed to embark on this journey in the first place. I want to especially thank some of my former flatmates in Hasselt—Ben, Lieselotte, Karolien, Mattias, Eveline, Bert, Brecht, Jolijn, and Anneleen—they made Belgium a second home for me, and I can consider them to be my *alternate family* now. All the chill-out evenings on the living room couch, parties, and (occasional) hangover-afternoons, gave me renewed energy to go back to my desk and continue working. I would also like to thank all the other friends I have made along the way, who have made life outside of work a pleasant experience.

Abstract

A user interface is the primary mean by which a user interacts with a computer. Interactive elements, placed on an interface, define the scope of interactions afforded to users. This thesis investigates placement issues central to the design of user interfaces. The primary goal is support the construction of user interfaces by improving or facilitating the placement of interactive elements on (1) graphical user interfaces (GUIs), and (2) post-WIMP user interfaces.

GUIs are the most-commonly used method for interacting with computers. They consists of interactive elements organised in a visual interface layout. Improving the construction of interface layouts positively impacts user performance and perception of the interface. However, objectively improving the placement of elements is non-trivial. The first part of my thesis addresses challenges towards *improving* placement on GUI layouts. To this end, I make two main contributions. In *Sketchplore*, I investigate design-time improvements by enabling interface designers to sketch and explore layouts using an interactive optimiser. In *Familiarisation*, I discuss a use-time approach to improve placement for individual users by applying principles of familiarity.

Post-WIMP interfaces go beyond the GUI paradigm, and open up new interaction possibilities. They support a larger set of interactive elements, such as sensors and actuators. Due to the added technical complexity, it can be challenging to place interactive elements onto such interfaces. The second part of this thesis focuses on facilitating the placement of interactivity onto post-WIMP interfaces, and makes two contributions towards addressing placement challenges. I investigate the placement of interactive electronic elements onto physical interfaces. I present *PaperPulse* as a tool for non-expert to place electronics onto paper interfaces, and extend the discussion to other physical interfaces such as wearables and smart home interfaces. In *BinPut*, I discuss the placement of standard input controls onto a diverse set of interfaces, and present a universal technique that can be applied to different types of input and devices.

The concepts and principles discussed in the thesis contribute towards addressing placement problems central to the construction of user interfaces. They can result in design interfaces that are performant, and that support a wide range of interactions. Quantitative and qualitative evaluations of the resulting tools and techniques provide evidence for the approaches presented in this dissertation.

Scientific Contributions

The contents of this thesis are a product of a set of academic publications presented at venues for dissemination of results in the field of HCI. The list of publications that contribute to this thesis are as follows:

1. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore layout designs with an optimiser. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 3780–3783. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890236
2. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplorer: A mixed-initiative tool for sketching and exploring interactive layout designs. In *Proceedings of the CHI '17 Workshop on Mixed-Initiative Creative Interfaces*. 2017
3. **KASHYAP TODI**, JUSSI JOKINEN, KRIS LUYTEN, and ANTTI OULASVIRTA. Familiarisation: Restructuring layouts with visual learning models. In *Proceedings of the 2018 ACM Conference on Interactive User Interfaces*, IUI '18. ACM, New York, NY, USA, 2018

4. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2457–2466. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702487
5. **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Enabling eyes-free interactions on jacket buttons. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI EA '14, pages 1549–1554. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2474-8. doi: 10.1145/2559206.2581155
6. **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Inconspicuous interactions on jacket buttons. In *Proceedings of the 2014 CHI Conference Workshop on Inconspicuous Interactions*, CHI EA '14. ACM, New York, NY, USA, 2014
7. **KASHYAP TODI**, KRIS LUYTEN, and ANDREW VANDE MOERE. Making smart homes personal: Fabrication and customisation of home interfaces. In *Proceedings of the CHI '15 Workshop on Smart for Life: Designing Smart Home Technologies that Evolve with Users*, CHI EA '15. 2015
8. **KASHYAP TODI**, DONALD DEGRAEN, BRENT BERGHMANS, AXEL FAES, MATTHIJS KAMINSKI, and KRIS LUYTEN. Purpose-centric appropriation of everyday objects as game controllers. In *Proceedings of the CHI '16 Extended Abstracts*, CHI EA '16, pages 2744–2750. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2892448

In addition to this, I have also presented some of my work as interactive exhibits and demos:

1. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore layout designs with an optimiser. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 3780–3783. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890236
2. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach to fabricating interactive paper. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 267–270. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3146-3. doi: 10.1145/2702613.2725430
3. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *SIGGRAPH 2015: Studio*, SIGGRAPH '15, pages 3:1–3:1. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3637-6. doi: 10.1145/2785585.2792694
4. BRENT BERGHMANS, AXEL FAES, MATTHIJS KAMINSKI, and **KASHYAP TODI**. Household survival: Immersive room-sized gaming using everyday objects as weapons. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 168–171. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890372

Collaboration Acknowledgements

While this dissertation has one author, the research conducted over the course of my thesis was completed in collaboration with other researchers. Several colleagues have made valuable contributions, and offered guidance and advice, without which the success of the various projects would not have been possible. My research trajectory was also heavily influenced by opportunities in the form of summer schools, internships, and collaborations. Here, I would like to acknowledge contributions of individuals to research presented in the thesis, and also identify my concrete contributions to the projects.

Chapter 2 The *Sketchplore* project was executed during an internship stay at the User Interfaces Group of Aalto University (Finland), under the guidance of Prof. Dr. Antti Oulasvirta. Dr. Daryl Weir contributed to the implementation of the predictive models and layout optimiser used in the system, and the quantitative user study. I was responsible for conceptualising, designing, and implementing the *Sketchplorer* design tool, and for the qualitative study with designers. This resulted in one full paper [146], an

interactive demonstration [145], and a workshop paper [147].

Chapter 3 The *Familiarisation* project was initiated by me during a second internship stay at the User Interfaces Group, under the guidance of Prof. Dr. Antti Oulasvirta. The project was continued after I returned to the Expertise Centre for Digital Media (EDM) at Hasselt University, and Prof. Dr. Kris Luyten advised and contributed to the completion of this research. Dr. Jussi Jokinen collaborated on this project, and was responsible for the implementation of the visual learning model. I was responsible for designing and developing the *Familiariser* system, and evaluating it with users. This research resulted in one full paper [141].

Chapter 4 The *PaperPulse* project was in collaboration with Prof. Dr. Raf Ramakers, under the guidance of our advisor Prof. Dr. Kris Luyten. The ideation and conceptualisation was a joint effort, and a result of several brainstorming sessions. A majority of the implementation of the PaperPulse design tool was done by Raf. I contributed to finalising the visual aspects of the GUI, design and implementation of the PaperPulse widgets, hardware-related challenges with circuit printing and electronics, and evaluating with end-users. This resulted in one full paper [118], two demonstrations [121, 119], and a poster [120]. The *Suit Up!* project was undertaken by me, under the guidance of Prof. Dr. Kris Luyten. Tom De Weyer provided help with hardware implementation of the interactive buttons. This resulted in one extended abstract [142] and one workshop paper [143].

The Evolving Smart Home project was conceptualised by me, un-

der the guidance of Prof. Dr. Kris Luyten and with the advice from my co-promotor Prof. Dr. Andrew Vande Moere. I proposed the Design–Deploy–Dispose (DDD) cycle, that formed the foundation of this research. Under my supervision, Jelco Adamczyk created the *EasyHome* software prototype to enable end-user home customisation. Steven Peeters continued working on the smart home theme, and under my supervision, implemented a logging toolkit for home interfaces. These two projects resulted in Jelco and Steven's Bachelor thesis publications, under the guidance of Prof. Dr. Kris Luyten. I proposed the initial ideas for these theses, assisted and advised during the implementation stages, and provided feedback on the textual content.

Chapter 5 The *BinPut* project was initiated by me, and conducted under the guidance of Prof. Dr. Kris Luyten and Prof. Dr. Johannes Schöning. I was responsible for the design, implementation, and evaluation of the input technique.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The design and construction of user interfaces is one of the main areas of focus in human–computer interaction. Entire academic conferences have been dedicated to this subject , and it has been one of the primary areas of interest for several research groups. The placement of interactive elements, and interactions, is a fundamental activity during the creation of any user interface. It entails adding or embedding elements onto the interface, positioning and sizing them, assigning various visual attributes, and specifying the interactions they enable. The placement of elements on a user interface defines the interaction possibilities enabled, and it is key to determining the usability and acceptance of the interface.

User interfaces can be categorised into: (1) Graphical user interfaces (GUIs) and (2) Post-WIMP user interfaces. Classically, GUIs are visual interfaces, rendered on a 2D display. Post-WIMP interfaces, inspired by GUIs, extend their capabilities beyond flat displays, and mouse-and-keyboard input. The main goal of this thesis is to improve

how elements are placed on typical GUI layouts, and to facilitate placement while constructing post-WIMP interfaces.

1.1 Interactive Elements for User Interfaces

The term *interactive element* is used to describe an elementary component of an interface, which enables some interaction. A user interface is composed of a set of such elements, combined to achieve a specific set of tasks. The exact definition of an interactive element is dependent on the type of user interface being addressed.

Graphical User Interface (GUI): Here, interactive elements are defined as visual interface elements, such as buttons, icons, input fields, and other commonly used widgets. These elements are typically placed or positioned on a canvas, and rendered on a two-dimensional graphical display. Users can visually locate these elements on an interface, and interact using input devices such as a keyboard or mouse.

Post-WIMP User Interface: Such interfaces contain an extended set of interactive elements. They typically consist of one or more interaction techniques that are not dependent on classical 2D elements known from GUIs [149]. Post-WIMP interfaces support different mediums and modalities for interaction, such as gestures, tangible interfaces, and ubiquitous interfaces. Interactive elements can be physical in nature, and can contain electronic components such sensors and actuators.

This thesis applies a design science approach to investigate challenges and opportunities related to placement of elements on user interfaces I present concepts, techniques, and artefacts to improve and

facilitate placement, and thus the design and construction of user interfaces.

1.2 Improving Placement in Graphical User Interfaces

The design process of constructing a GUI involves placement of elements on a two-dimensional canvas. The resulting composition of elements is referred to as an *interface layout*. The placement and organisation of interface elements on a graphical layout influence the overall quality. Aspects such as position, size, colour, among others, determine both user performance and overall perception of the interface. Poor placement of elements can lead to minor user annoyances at best, and potential disasters at worst (e.g. [138]. In 'The Design of Everyday Things' [106], Norman mentions six principles for good user interface design. Placement of elements has direct influence on some of these principles such as visibility and consistency. Gestalt principles [157] underline the importance of placement aspects such as positioning and sizing of elements. Predictive models such as Fitts' Law [40] can quantify the effect of placement on user performance, and have been used extensively to design and validate user interfaces. It is evident that effective layout design is crucial for the success of an interface. A well-designed layout should pay attention to visuospatial aspects such as effort required to search for key features, pointing time to select elements, and learning aspects that aid in recall. It should also take into account aesthetic details such as colour harmony, balance,

and grid alignments. The first part of this thesis, therefore, focuses on improving the placement of interactive GUI elements on visual layouts.

Advances in interface design tools and technologies have greatly facilitated the process of layout creation. Prototyping, sketching, and design tools enable designers to externalise their ideas, and find solutions to design problems. Extensive GUI programming frameworks, interface builders, and toolkits, have enabled programmers to convert design prototypes into functional user interfaces. Designers possess the knowledge and skills to design good interfaces, but can often be biased by their experience, thus restricting their exploration of the entire solution space [32]. Additionally, they might not have complete information about the target users, and lack methods to objectively verify the usability of their produced designs. While commercial design tools for interface creation are readily available, they often provide only limited assistance, or guidance, during the design process. They tend to take a neutral stance towards design, and refrain from steering designers towards better outcomes. Brad Myers et al. [99] discuss how tools could instead "enforce or encourage" highly usable interfaces. In this spirit, we believe that it can be beneficial to support a more comprehensive design exploration, at early stages, enabling designers to detect objectively good alternatives, and improve their designs. This inspires the first research question in this dissertation:

Research Question 1: *How can we computationally support designers in the process of design exploration during early stages of placement of interactive elements on a graphical interface?*

In chapter 2, I address challenges for enabling systematic design

exploration, and suggest *Sketchploration* as a viable technique. Sketchploration combines early-stage sketching with design exploration with the aid of a layout optimiser. It enables designers to quickly create possible solutions to a design problem. The machine abstracts the design problem from drawn sketches, and uses this to generate novel design alternatives, and proposes these to the designer in the form of suggestions. By using predictive models, the optimiser can select objectively good designs, thus improving the overall quality of solutions.

Sketchploration enables designers to improve designs for the general population. Individual users often encounter a wide diversity of designs. Placement of elements can vary greatly from one interface to another. As a result, users are required to adapt to each interface, and expend additional effort in learning different strategies. In addition to designer-enforced *global* consistency, it can be beneficial to also support *local* consistency in placement of elements for each user. This motivates the next research question:

Research Question 2: *How can we adapt graphical interfaces for individual users' by automatically placing elements at familiar locations, at use-time, such that they are consistent with a user's mental model and enable faster visual recall?*

The main challenge is to derive an accurate representation of the user's mental model, and use this to place elements on a new interface. In chapter chapter 3, I explore *Familiarisation* models, and propose techniques to automatically place elements on new interfaces.

By addressing both design-time and use-time placement of interactive elements, the first part of this thesis addresses overall improvements in GUI layouts.

1.3 Facilitating Placement in Post-WIMP User Interfaces

Mark Weiser's vision of ubiquitous computing [156] stressed upon the interweaving of hardware and software such that computers themselves would disappear into the background, and computation would be seamlessly embedded in our environments. Post-WIMP interfaces are a step towards realising this vision, and expand the ways in which we can interact with computers. As a result of these added capabilities, post-WIMP interfaces tend to be require more engineering than GUIs. In GUIs, placement of elements deals with positioning them on a digital canvas, and assigning them visual and interaction attributes. In the context of physical interfaces, placement additionally involves the integration (or embedding) of other interactive components, such as sensors and actuators, into physical mediums. Placing various interactive elements on different post-WIMP interfaces is non-trivial and technically challenging. The second part of this thesis, therefore, aims to facilitate the placement and integration of interactive elements onto such interfaces, to enable various interactions. Unlike general-purpose GUIs, post-WIMP technologies support the creation of special-purpose user interfaces, customised towards specific tasks and interactions. These open up opportunities for highly-customised and personalised interfaces. Given the personal nature of such interfaces, it is desirable for end-users to create and adapt the interface by placing interactive elements on them. However, non-expert users lack technical skills and knowledge required for creating such interfaces. This motivates the following research question:

Research Question 3: *How can we facilitate non-experts in placing interactive electronic elements on special-purpose physical interfaces by obviating the needs for programming and electronics skills?*

In chapter 4, I discuss an integrated workflow to create physical, paper-based, interactive interfaces. *PaperPulse* is a design tool that enables -experts to place low-cost electronic widgets onto paper substrates, and specify the desired interactivity, without requiring them to acquire expertise in programming or electronics. The end-to-end approach guides the user through the process of starting from a visual design, and creating a standalone interactive artefact. I also briefly discuss facilitating placement of electronics on other physical mediums such as clothing and home interfaces.

An end-to-end workflow enables the creation of special-purpose interfaces, supporting customised interactions. However, it does not address traditional input tasks such as navigation, selection, or text and number entry. On GUIs, such tasks can be efficiently executed using a mouse and keyboard. In contrast, post-WIMP interfaces typically integrate various sensing mechanisms to present users with alternate interaction techniques for input. As a consequence, input data from sensors can be error-prone or of low granularity. This issue inspires the final research question of this thesis:

Research Question 4: *How can we facilitate the placement of standard input controls on post-WIMP interfaces while maximising consistency across interfaces and reducing re-learning of the input technique?*

Chapter 5 addresses challenges for placing consistent input controls on a diverse set of interfaces. It discusses a universal input technique *BinPut*, which adapts the binary search algorithm to enable con-

Interface Type	Research Question	Contribution
Graphical User Interfaces (Part I)	*Improve interface layouts by supporting design-time exploration*	Sketchplore (Chapter 2)
	Improve interface layouts automatically to individual users	Familiarisation (Chapter 3)
Post-WIMP User Interfaces (Part II)	*Facilitate non-experts to create special-purpose interfaces*	PaperPulse (Chapter 4)
	Facilitate placement of standard input controls across interfaces	BinPut (Chapter 5)

Table 1.1: The four research questions for improving and facilitating placement on user interfaces, and the corresponding contributions presented in this thesis.

sistent input across different types of post-WIMP interfaces. The technique has minimal input requirements, and can be applied to any inherently-ordered input type. Thus, BinPut provides a simple technique for placing input controls on interfaces where it might otherwise be tedious or infeasible.

The above four research questions cover key aspects towards the overall goal of improving and facilitating the creation of user interfaces. Table 1.1 provides a summary of the research questions and corresponding contributions.

1.4 Chapter Overview and Contributions

This dissertation makes contributions in the area of (1)improving and (2) facilitating the placement of interface elements and interactions on user interfaces.

First, it investigates methods to improve the quality of graphical user interfaces, by allowing designers to explore optimised layout designs, and by enabling automatic use-time repositioning of elements. Next, it discusses facilitating the creation of customised post-WIMP user interfaces, and placement of input controls on such interfaces. The core chapters (2–5) are a collection of concepts, projects, and discourses, which contribute towards the broader goal of supporting the construction of user interfaces. Below is a brief overview of the these chapters, highlighting the main contributions.

Part I: Improving Placement on Graphical UIs

Chapter 2 discusses improvements to design-time placement

on graphical interfaces and presents *Sketchplore*. It enables designers to improve placement of interactive elements on graphical layouts during the ideation stage of the design process. *Sketchplorer* uses a mixed-initiative approach to provide optimised layout suggestions, enabling rapid generation of diverse solutions to a design problem.

Chapter 3 addresses use-time improvement to placement of elements. It presents principles of *familiarity*, based on visual learning models, that can improve the placement of layout elements on new interfaces for individual users. *Familiariser* automatically positions elements on new and unvisited interfaces to improve them for individual users.

Part II: Facilitating Placement on Post-WIMP UIs

Chapter 4 investigates placement of electronics on physical interfaces. It presents *PaperPulse*, providing non-experts with a simplified workflow for placing interactive elements on paper substrates. The system automates circuit generation, and enables non-experts to specify interactions with a visual programming technique. It extends the concept beyond paper to other interfaces, such as wearables and smart homes.

Chapter 5 tackles placement of input controls onto physical and graphical interfaces, and presents *BinPut* as a universal technique that enables this for a diverse range of interfaces and modalities, and for different types of input.

PART I

Improving Placement on Graphical User Interfaces

A graphical user interface (GUI) contains interactive visual elements, such as windows, icons, menus, pointers (WIMP), rendered on a graphical display. GUIs are the most commonly-used type of interfaces, enabling us to efficiently and intuitively communicate with a computer. In the first part of this dissertation, I highlight the importance of placement of interactive visuals elements on GUIs, and investigate concepts, techniques, and tools to improve the placement of interactive elements on graphical layouts.

Chapter 2

Improving Design-Time Placement on Graphical Layouts

Interface designers create graphical user interface (GUI) layouts by placing a set of interactive elements on the interface canvas. They make placement decisions such as position, sizing, and colour of elements, and the overall organisation of the layout. Sketching is often used as a problem-solving technique for quickly generating plausible solutions. For interactive digital layout designs, such as websites, the design space of possible solutions is immense, making it hard for designers to explore a large number of ideas in a short timespan.

In this chapter, I address this design-time exploration challenge, and proposes *sketchploration* as a concept to improve design-time placement of elements on GUIs. *Sketchplorer* combines sketching with design exploration using an optimiser. As the designers sketches out viable designs on the canvas, the optimiser abstracts the design task, and generates new designs optimised for performance and aesthetics.

Designers can then use this to quickly create viable alternatives that explore a larger part of the design space than otherwise possible.

The contents of this chapter are based on the following publications:

1. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, DIS '16, pages 543–555. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4031-1. doi: 10.1145/2901790.2901817
2. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore layout designs with an optimiser. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 3780–3783. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890236
3. **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplorer: A mixed-initiative tool for sketching and exploring interactive layout designs. In *Proceedings of the CHI '17 Workshop on Mixed-Initiative Creative Interfaces*. 2017

2.1 Introduction

A graphical user interface (GUI) typically consists of interactive elements, such as buttons, widgets, input fields, and other visual and textual contents. Designers create GUI layouts by placing a set of these interactive elements, based on the design task, on a two-dimensional

canvas. They make placement decisions such as position, sizing, and colour of elements, and the overall organisation of the layout. The aim of the designer while constructing a layout is twofold—the interface should be visually appealing to attract users, and should enable users to efficiently complete desired tasks. This chapter is motivated by the observation that optimisation methods have great untapped potential in design tools for constructing such layouts. It focuses on the activity of *sketching graphical layouts*, in which a designer applies a visual problem-solving approach for placing elements on a canvas.

2.1.1 The Placement Problem in GUI Layouts

Placement of UI elements on GUIs can be a challenging task for designers. The aim is to create a meaningful and performant layout, for a given design task, by placing interface elements on a given canvas. We define the placement problem as an interface layout problem that includes making decisions related to:

- **Position:** The (x,y) location of elements on the canvas.
- **Size:** The width and height of each element.
- **Colour:** Colours assigned to the element, and the overall colour harmony of the design.
- **Organisation:** Organisation of elements on the canvas to maintain properties such as visual balance and grid quality.

These are key aspects that influence usability of resulting designs, and thus important for designers to correctly estimate. In his book

on *Sketching User Experiences* [23], Buxton gives a background on the role of sketching during the design of interfaces. Sketching is used by designers to find possible solutions by quickly placing elements on a canvas. Factors such as experience bias and lack of knowledge about the users can, however, limit this design exploration. Objectively estimating the impact of changes to a layout on the overall usability and performance is also hard for designers. Predictive models can evaluate a design to estimate factors such as the time required to visually find an element (visual search), and movement time for manipulating a cursor to acquire targets, among others. Optimisation techniques can be used to quickly generate designs by placing elements on the canvas, and evaluate designs for aesthetics and performance. The fluidity of sketching, combined with this speed and objectivity of optimisation, can offer a promising approach to enabling systematic exploration, and solving placement problems, in early stages of layout design.

2.1.2 Sketching vs. Optimisation

Sketching and optimisation are typically seen as two opposing activities. On the one hand, sketching is known to be an imprecise and fluid exercise, where the designer attempts to generate several feasible design solutions for a given problem. During this process, the designer leaves ample room for uncertainty and ambiguity. To this extent, sketching can be defined as a *visual tool for problem solving*.

On the other hand, optimisation is a precise search for the single best solution to a given problem. The optimisation process strives to be unbiased and objective, and does not leave room for much uncertainty.

However, the two activities can also be thought of two ways of reaching the same goal. Both sketching and optimisation can be eventually framed as accelerated search processes to quickly solve a given problem. They attempt to explore the design space efficiently, filter out infeasible solutions gradually, and finally arrive at one, or multiple, good solutions.

2.1.3 Research Question

The observation that an interplay between sketching and optimisation can aid designers to quickly generate multiple good solutions to a design problem inspired the following research question:

*How can we computationally support designers in the process of design **exploration** during early stages of placement of interactive elements on a graphical interface?*

The remainder of this chapter describes, in detail, our approach to tackling several relevant research challenges to enable this, and presents *Sketchplore*, a concept and tool for designers to simultaneously improve their designs and systematically explore design alternatives.

2.1.4 Sketchplore: Sketching and Exploring Layout Designs

From a combinatorial perspective, the design of layouts is notoriously hard. Take, for example, a canvas of 1024×768 pixels [1] , divided into a 24×32 grid. Here, there are 158,400 one-element layouts and a whop-

[1] Standard XGA resolution has 1024×768 pixels.

ping 10^{41} eight-element layouts. Although algorithms may not be able to find the optimal solution in such large search spaces, they can "parallelise" search, and find candidate solutions and suggest them to designers. This could help designers in exploration, who are known to be limited to a handful of designs per iteration [32]. Also, algorithms can complement designers by exploring design spaces neutrally without being constrained by past experiences, to produce designs that the designer might not otherwise conceive. Employing an optimiser might also improve the quality of designs for end-users (see [43, 111, 167]). A combination of sketching and optimisation, therefore, could enable designers to improve placement of elements on GUIs.

However, several hard research challenges emerge. First, layout design is a complex, multi-objective task addressing not only usability but also aesthetic qualities [58, 164]. Presently no algorithmic approach exists that can address both. Second, optimisation typically takes a long time, due to combinatorial complexity, and no solution has been shown for fast-paced, iterative design of layouts. Third, although optimisation methods can attack very complex design problems, their insistence on *precise* inputs contradicts sketching. To better fit with design practice, optimisers should radically reduce the effort for defining tasks (see e.g., [9]). Crucially, they should not attempt to override design thinking. Instead, final decisions should be left to designers, who possess knowledge that computers might not.

'Sketchploration' is a novel concept to exploit interactive optimisation methods in design tools, in particular for layout sketching. We approach sketching as a *problem-solving* activity and a tool for *visual thinking* [23, 101]. The goal is to enable access to results of real-time optimi-

sation, yet impose as few control requirements on the designer as possible, in order to support the natural flow of sketching. To achieve this, Sketchplorer, illustrated in Figure 2.1, relaxes typical requirements for controlling an optimiser in a way that respects the design process. Unlike with previous solutions (e.g., [43, 9]), no additional input is required from the designer. As the designer sketches, the system infers the designer's task automatically. The optimiser simultaneously searches for local improvements (small changes), and explores global alternatives (large changes). Importantly, it deploys several predictive models of user performance and perception adapted from literature. This allows it to make informed suggestions that "pull" the designer toward usable and aesthetic designs. Sketchplorer does not override the designers' sketch, but presents optimised designs as glanceable suggestions. Technically, sketchploration extends *model-based interface optimisation* [9, 43, 167] to real-time design exploration under ill-specified and changing design goals.

This chapter presents the first investigation of the concept, focusing on *interactive layouts* familiar from GUIs, web pages, menus, and dialogs. It complements existing work by showing how an interactive optimiser can be integrated with fast-paced and unconstrained early-stage sketching. Our implementation presently supports 10 common types of elements and hierarchical (nested) placement. The concept could be implemented for any sketching tool that affords extending the workspace and communication with a server. This chapter also reports results from two empirical evaluations—with end-users and with trained designers.

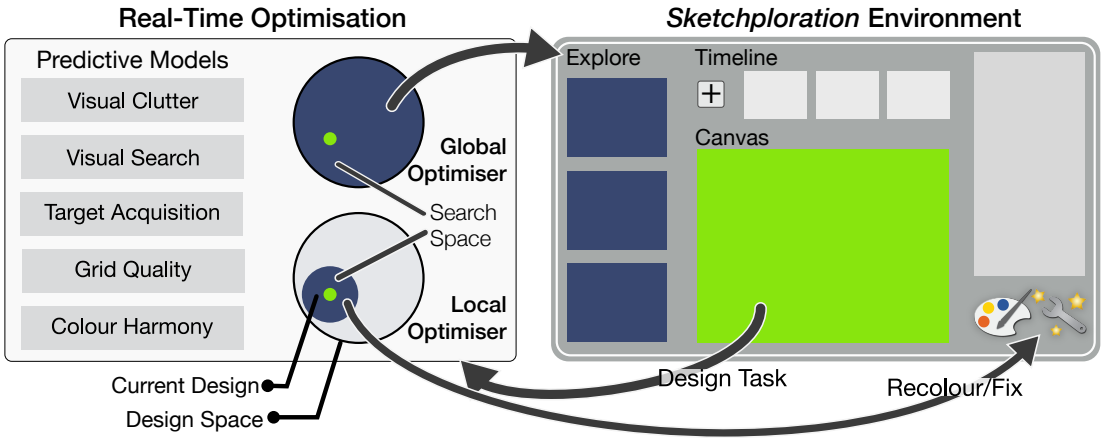


Figure 2.1: *Sketchplorer* is an interactive layout sketching tool supported by real-time model-based optimisation. The tool is designed to facilitate the creative and problem-solving aspects of sketching without requiring extensive input. While a designer is sketching, a design task is automatically inferred. The optimiser uses predictive models to make suggestions for local and global changes that improve usability and aesthetics. Suggestions appear on the side, and never override the designer's work.

2.2 Background: Sketching Tools and UI Optimisation

Our goal is to support sketching with computational methods. In this work, we investigate sketching as a tool for visual thinking and problem-solving, and do not delve into freehand drawing and rendering aspects of sketches. When understood as problem-solving, sketching is characterised by its quick, ambiguous, and uncertain nature [50, 162]. The goal of designers is not a point design, but rather exploration. Sketching unfolds as an iterative process of idea-generation, refinement (exploitation), and redefinition. Designers entertain multiple hypotheses and may backtrack to previous designs. Details to a sketch are added gradually, and they can be changed at any stage. Idea-generation in sketching has recognised limits and biases [32]. These properties make sketching both a challenge and an opportunity for computational support.

2.2.1 Sketching Tools and Interaction Techniques

Research on computational support for sketching originates from Sutherland's Sketchpad [137], which highlighted the benefits of a digital medium. Johnson et al. [68] offer a comprehensive review of computation support for sketching. Here, we highlight a few main trends in research.

First, improved recognition technologies have brought pen-and-paper like techniques to sketching [161]. Second, several interaction techniques have been proposed to enhance the drawing of shapes in

sketching (e.g., [8]). Third, some approaches have looked at integration with other activities in design, and a better support for going back and forth between designs. SILK [83] and DENIM [88] explored several such techniques. Fourth, sketching has been extended from 2D spatial displays. For example, Kitty [73] enabled sketching for animations and dynamic authoring of illustrations. Fifth, design heuristics have been implemented in sketching tools. Examples include colour palettes [97], template-based sketching [70], and design guidelines. While these ensure that designs meet certain standards, they allow neither exploring designs nor refining them for some objectives.

2.2.2 Heuristic and Data-Driven Methods for Layout Generation

Heuristic approaches to layout generation have explored balance, consistency, or the golden cut (e.g. [39, 90]). Although heuristics can produce results that are visually appealing and resemble real designs, they do not predict effects on end-users. They lack means for conflict resolution. To our knowledge, there are no heuristic approaches that solve both spatial and visual aspects of layout.

Data-driven approaches such as Webzeitgeist [81] mine a large number of designs and can produce new ones for designer given inputs. Although colour, size, position, and grouping can be addressed, the approach results in basically "mimicry" of existing designs. It does not offer a principled way of setting objectives for goals like usability and aesthetics. They offer no guarantee that the outcomes are good beyond visual appeal.

Some work has been done on automatic generation and improvement of static graphical media such as posters, flyers, or slides. O'Donovan et al. [107] used an energy-based model considering aesthetic heuristics, such as alignment, balance, and flow. DesignScape [108] is a tool for assisting novice designers in creating graphical media. While our work, at first glance, resembles DesignScape, the assumptions and underlying mechanisms of the two systems are very different. DesignScape does not consider interactive layouts, which form the basis of GUIs. Additionally, the tool is meant to improve final-stage designs, based on realistic content. It is not designed for the uncertain and ambiguous nature of initial-stage sketching, where the actual content is still subject to change. With regard to underlying optimisation mechanisms, DesignScape uses a reduced design space which does not include element colour, and makes simplifying assumptions about aesthetics rather than using validated predictive models from psychology.

Finally, there has been extensive work on automating layout generation based on constraint satisfaction. Interfaces are specified as a set of semantic and spatial constraints, and solver programs generate possible designs for given screen sizes. Lok and Feiner [89] reviews these techniques. However, the constraints still have to be constructed by a designer, and moreover the generated layouts have no guarantees about aesthetic quality or interaction performance.

2.2.3 Metrics and Model-based Optimisation

Previous works have evaluated interfaces automatically using different metrics. BaLOReS [91] proposed five structure principles and metrics to guide designers and evaluate resulting designs. QUESTIM [166] semi-automatically evaluated website layouts using several metrics, and displayed the results to designers. DesignEye [124] used models of saliency and visual clutter to aid design teams in evaluating and improving designs. While these works aimed at evaluating an interface layout, they did not investigate systematically suggesting improvements.

Model-based user interface optimisation uses combinatorial optimisation to automate interface generation. The idea is to represent a design problem and design knowledge (e.g., user simulations, models, heuristics) as an objective function for a search algorithm that iteratively improves designs for the stated objectives. Unlike heuristic approaches, model-based optimisation relies on theories and predictive models of how users interact or perceive a layout and an algorithm that searches the design space systematically. The layout problem can be considered as an instance of the well-known assignment problem [80]. Fitts' law has previously been used, together with bigram data on transition frequency, to find keyboard layouts that minimise expected finger travel distance [87]. This idea has been extended to widget layouts in SUPPLE using branch-and-bound [43].

Some previous works have investigated model-based optimisation of interfaces with a designer-in-the-loop. DON [77] assisted designers in generating menus and dialog boxes using an integrated knowledge base model. TRIDENT [152] enabled designers to generate business-

oriented application interfaces. It encapsulated design guidelines in a decision tree, and suggested the best path for a given task. It allowed designers to intervene and choose alternate paths, to guide the design process. Closer to the work presented in this chapter, MenuOptimizer [9] *interactive* is a design tool that used predictive models of human performance for layout optimisation. It used a model of menu search together with a consistency heuristic to optimise hierarchical menus for application. It integrated several types of support to the QtDesigner development environment. It introduced an interactive optimiser that proposed global and local changes like moving a menu item to improve user performance. However, it was designed for "point optimisation" and contained an overwhelming amount of controls and visualisations and insisted on problem specification for the optimiser. The system was also limited to hierarchical menus, and did not consider visual designs. The authors concluded that designers mostly used global suggestions (suggestions for the whole menu system).

In this chapter, we extend model-based optimisation to design tools for visual layouts, thus enabling designers to sketch and explore placement of interactive elements on graphical interfaces.

2.3 Walkthrough and Design Overview

We had four objectives for integrating computational support to a design tool:

1. Support quick and ambiguous sketching, and leave room for uncertainty, by providing capabilities to defer the task of specifying

details at any stage of the design process.
2. Allow fluent shifts between exploration and refinement.
3. Provide support for multiple hypotheses, and give an overview of progress, by providing a non-destructive, and editable, timeline of previous alternate designs.
4. Minimise user actions not related to the design activity itself by inferring details whenever possible.
5. Eliminate ambiguity, in designers' minds, while perceiving optimised results by communicating with the designer in a timely and predictable manner.

2.3.1 Walkthrough: Designing a Blog Page

This walkthrough illustrates the use of Sketchplorer from a designer's perspective.

Sketching the initial layout: Sketchplorer initially presents the designer with an empty canvas. The designer starts sketching by creating a structure for her design (Figure 2.2a). Drawn elements can be moved around, or resized, at any time. She ambiguously sketches out boxes, serving as proxies for the elements of her blog page. Sketchplorer dynamically infers hierarchy of elements, and does not require the designer to explicitly specify ordering or grouping. It starts computing both local and global suggestions in the background.

Refining and adding details: An *inspector* panel, similar to most commercial sketching tools, sits on the right-edge of the display, and can be pulled out at any time. This can be used to specify details, such as the element type, colour, and importance (usage probability)

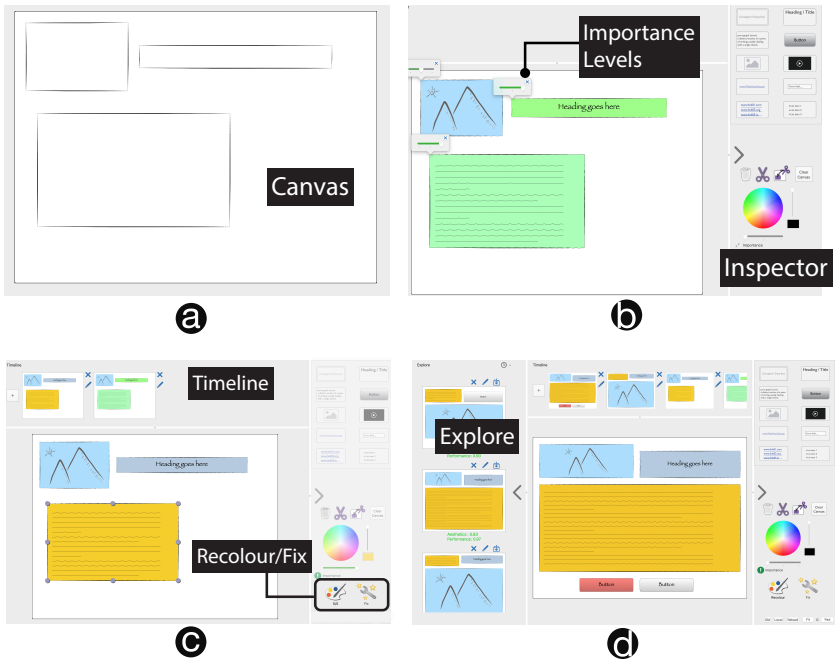


Figure 2.2: Overview of Sketchplorer. A large canvas is surrounded by the various features. Designers can (a) ambiguously sketch layout designs, and (b) refine and add details. They have immediate access to a timeline of saved designs. They can improve designs by (c) using *fix* and *recolour* suggestions, or (d) exploring globally optimised designs.

of objects on the canvas (Figure 2.2b). The designer now adds details to some of the elements. For instance, she indicates that her blog page has a heading and a paragraph element, and marks them as being of high importance for the optimiser. She also specifies that the image (site logo) is of low importance. Satisfied with the first version of the design, she taps the save ('+') button. This adds the current design to the designer's *timeline*, and provides a preview. The designs here can

be retrieved, and edited, at any later time. This enables the designer to see the evolution in designs, borrow previous ideas, and select feasible alternatives.

Fine-tuning and local changes: While the designer sketches and refines, the system continuously streams the description of the current design to the optimiser. The local optimiser uses the current design as a starting point to suggest fixes and provide recolour options. Pulsating icons in the inspector panel appear when *fix* and *recolour* options are available (Figure 2.2c). The designer refers to these suggestions. She realises that by using the recolouring suggestion, she can make the paragraph of text stand out. She chooses one of the recoloured layouts, and continues working on her sketch.

Exploring new designs: By abstracting from the current design, the global optimiser retrieves unique designs, and returns them to the designer. An *explore* panel, residing on the left edge of the display (Figure 2.2d), is periodically updated with these designs. The designer browses through the list of alternatives, and finds two interesting alternatives. She adds the first to her list of saved designs, and drags the second onto the canvas, to continue working on it.

In a short duration, the designer's collection of *saved layouts* is populated with several feasible alternatives—some sketched by the designer, and the others with the aid of the optimiser. While the above phases appear to be linear, in practice, sketching and exploration phases are intertwined.

2.3.2 Overview of Interactions

Sketchplorer is designed for a multitouch environment using a large display, and uses touch gestures for all controls.

Sketching and refinement: Sketchplorer allows designers to either sketch ambiguous bounding boxes for layout elements, or pick out a specific element type and draw it on the canvas. It takes care to accurately order every element on the canvas, without requiring the user to explicitly *bring-to-front* or *send-to-back*. Each time an element is changed, so as to change the hierarchy, the ordering is dynamically adapted. Hierarchical groups of elements can be selected and manipulated at the same time. This inferred hierarchy is also essential for the internal representation of a layout in the optimiser. The colour picker allows designers to select the hue–saturation combination, and adjust the brightness. Double-tapping on an element reveals an in-place popover, and allows adding details without having to move to the inspector. Individual elements' importance can be adjusted in the inspector panel. Alternatively, an overlay can be enabled, that displays the importance of each element, and allows batch adjustments (Figure 2.2b).

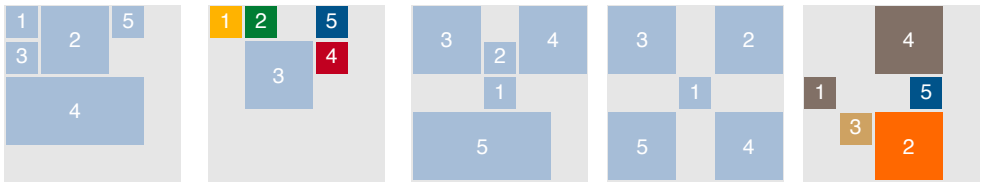
Saved versions and timeline: Benefits of interactive timelines and alternate versions have been emphasised in HCI literature [139]. In Sketchplorer, the current design can be added to the timeline at any time. The timeline provides an overview of all saved designs, and any alternative design, or intermediate sketch, can be dragged back to canvas. This allows designers to compare designs, have an overview of the evolution of their designs, and iterate over sketches.

Integration with the optimiser: To the designer, the optimisation appears as a two-pronged approach, consisting of local and global

optimisation. The local optimiser listens in on every change in the design, including changes in sizes and positions of elements. It creates fine-tuned designs, that maintain the overall composition of the original sketch, but improve certain aspects of it. Recolouring suggestions maintain the sizing and placement of elements, and offer harmonious recolouring suggestions, which also improve aesthetics and performance. In contrast, the global optimiser listens exclusively to changes in design tasks, and acts upon them. It abstracts away from exact details, allowing it to explore the entire design space, and generate unique designs. It performs exploration in real-time and periodically returns improved results that are immediately displayed to the designer in an expandable *explore* panel. The optimiser focuses on creating unique and improved solutions, and not on simply refining or polishing a solution.

2.4 Predictive Models for Interactive Layouts

Sketchplorer is the first user interface optimiser using models of human performance and perception to optimise both spatial and colour aspects of interactive layouts. We chose models that cover both aesthetic features, such as symmetry, and sensorimotor performance measures, such as target selection time. The models have also been validated in empirical studies and shown to align with user preferences and performance. In this way, we aim to produce layouts that are aesthetically pleasing and usable in a predictable way. However, note that the choice of models in the system is flexible. If there is evidence for another model being better than those we use currently, it is very



(a) Clutter **(b)** Visual search **(c)** Target acquisition **(d)** Grid quality **(e)** Colour harmony

Figure 2.3: Results for the colour patches task. These five layouts are outcomes when optimising for a *single objective at a time*. Sketchplorer combines these objectives into multi-objective search. Numbers refer to a ranking based on frequency of use.

simple to add this without changing the workflow of Sketchplorer.

2.4.1 Overview: The Colour Patches Task

In order to examine the effect of optimising for a model as an objective, and learn how the models affect optimisation, we created the *colour patches task*. Here, the optimiser's goal is to assign five rectangular elements to a 5×5 grid. Each element is a block or patch with a single colour. We assume that each patch is interactive, with a certain probability of being targeted. We sample these probabilities from a Zipf distribution [168]. These probabilities are used to weight the average visual search and target acquisition times using a formula given later in the chapter. To produce candidate designs, we traverse the elements in a random order and pick a random colour, size and position for each. The colours are chosen from Kelly's set of perceptually distinct colours [74], and the sizes are chosen randomly from three fixed sizes. For each objective, we evaluate many random designs to find the one

with the lowest objective value, then perform small changes to search for local improvements in the design.

Figure 2.3 shows the results. Elements are numbered by their usage probability, with 1 being the most common. The results show that each objective focuses on either spatial or colouring aspects, but may ignore or contradict the others. In the following sections, we will detail these results.

2.4.2 Visual Clutter

We use the Rosenholtz model [125] to minimise visual clutter. The idea is that as more objects are added to a display, it becomes more difficult to place a new object that it is perceptually unique and easy to identify. The layout elements have some distribution of visual features. As the number of elements grows, the feature distribution occupies more of the available space. This model has been shown to correlate with user perception of clutter, a correlate of aesthetic preference.

Given vectors of the features for each object on the display, we compute the mean and covariance Σ of these vectors. In our implementation, we consider only colour in our feature vector. The clutter of the display is then defined simply as the determinant of the covariance matrix, $|\Sigma|$. This can be thought of as the volume of the 1 s.d. covariance ellipsoid. We take the inverse of the clutter score for the objective function.

Figure a shows the colour patch results for this model. Each item has the same colour, which makes sense as this choice minimises the volume of the colour covariance.

2.4.3 Visual Search

We implement the Kieras-Hornof model of visual attention [76] to maximise visual search performance by enhancing the perceptual uniqueness of commonly searched elements.

Prediction consists of two parts: (1) a set of availability functions determine whether the features of a target are perceivable from the user's current eye location; and (2) a simple GOMS-style algorithm estimates the time in milliseconds required to locate the target.The availability functions are based on the eccentricity from the current eye location e and angular size s of the target. Additive Gaussian random noise with variance $\sigma^2 = vs$ proportional to the size of the target is assumed. For each feature, a threshold t is computed as $t = ae^2 + be + c$ and the probability that the feature is perceivable is given by:

$$P(\text{available}) = 1 - \Phi\left(\frac{t-s}{\sigma^2}\right), \tag{2.1}$$

where $\Phi(x)$ is the c.d.f. of the standard normal. We further define $P(\text{available}) = 1$ when $e < 1°$, since targets in the fovea are always perceived.

We use parameter values from the original paper, which were shown to accurately predict visual search time for data from an older study [158]. Note that our implementation only uses colour as an availability feature, since the authors found size and shape could only reliably be perceived for targets close to the fixation location.

For memory effects we use a simple logarithmic model based on the number of previous acquisitions. We optimise assuming the user is an expert and has visually searched the interface 1000 times in to-

tal. However, this assumption can be easily changed for example to account for novices.

Figure b shows the visual search colour patches. Each item has a different colour to make it visually unique, and all items are clustered in the top left of the grid, where the simulated search begins. More important items are closer to the left corner. Note that this objective conflicts with the clutter metric, which rewards layouts where everything is the same colour. Thus, there is a tradeoff between aesthetics and visual search performance that can be controlled by adjusting the weights.

2.4.4 Target Acquisition

Following previous work on performance-optimisation of layouts [87, 167], we deploy a variant of Fitts' law [94] as our model of target acquisition. Fitts' law favours transitions between large elements at minimum distances, and estimates the upper bound of expert pointing performance. Including this model allows us to optimise for efficient selection of elements, rather than purely for aesthetics. T is the time it takes for an end-effector to reach a target from a given distance D. For target width W, T is given by:

$$T = \sum_{r \in R} t_r = \sum_{r \in R} \left[a + b \log_2 \left(\frac{D}{W} + 1 \right) \right] \tag{2.2}$$

where R is the set of responses when navigating to the target.

When optimising, we assume that users interact with the interface elements using touch. We use values of a and b from a recent study

[36]. To compute D, we assume that the starting point of the finger is the centre of the display. We compute W according to the angle of approach. Fitts' law parameters for other input devices, such as the mouse, are readily available, and the optimiser can easily be adapted for these devices.

Figure c shows the colour patches for target acquistion. The most important element is centrally placed, close to the assumed starting position. The other elements are arranged around the center and have larger sizes so that they can be acquired without moving the finger far from the center.

2.4.5 Grid Quality

We use the Balinsky symmetry metric [10] as a measure of grid quality. It calculates the distance to the closest symmetrical layout. It was shown to correlate with aesthetic preferences.

We start with the layout vertex set $s = \{(x_1, y_1), \ldots, (x_n, y_n)\}$, rescaled such that the x-axis coincides with the axis of symmetry and the y-axis contains the centre of mass of the x_i. This is mapped to the complex plane to obtain a set $z_i = x_i + Iy_i$. These points are horizontally symmetrical if they consist only of real values and complex conjugate pairs. A fundamental result in complex analysis states that this is equivalent to the stipulation that all coefficients a_j of the polynomial

$$P_n(z) = \prod_{j=1}^{n}(z - z_j) = \sum_{j=0}^{n} a_j z^j \tag{2.3}$$

are real. Thus, the asymmetry for a layout is obtained by constructing

this polynomial and averaging the size of the imaginary parts of the coefficients. Vertical symmetry is scored in the same way, after a simple coordinate transformation. Our overall metric is an even weighting of these two scores.

Figure d shows the colour patches for this metric. As expected, the result is symmetric in both axes.

2.4.6 Colour Harmony

Harmonic colours are sets of colours that combine to provide a pleasant visual perception. Harmony is determined by relative position in colour space. There is no universal definition for a harmonic set. Here, we use the colour templates that were proposed by Cohen-Or et al. [31]. These consist of one or two sectors of the hue wheel, with given angular sizes. These templates can be arbitrarily rotated to create new sets.

The distance of a layout X from a template T is given by

$$D(X,T) = \sum_{e \in X} DH(e,T)S(e), \tag{2.4}$$

where $DH(e,T)$ is the arc-length distance between the hue of element e and the hue of the closest sector border in T, and $S(e)$ is the saturation channel of e. If a colour falls inside one of the sectors of T, $DH(e,T)$ is identically zero.

In order to rapidly evaluate many layouts, we pre-generate a set C of 78 harmonic sets by rotating the templates in fixed increments. We

define the colour harmony H by:

$$H(X) = \min_{T \in C} D(X, T) \tag{2.5}$$

Figure e shows the colour patches with the best match to one of the harmonic sets. Patches 1–4 have similar hues but different saturations, and patch 5 has a hue on the opposite side of the colour wheel. Together these colours form a harmonic set with the Y-template defined in [31]. Note that the placement and ordering of the patches is somewhat chaotic, since this model does not consider those factors.

2.4.7 Scope and Limitations

These models allow us to deal with the positions, size, alignment and colour of layout elements. An optimiser is able to search for layouts that improve for user performance, but also for aesthetic qualities. If implemented alone, each objective drives a layouts to an unbalanced design favouring one aspect. A multi-objective optimiser allows finding good compromises. A limitation is that these models do not allow us to deal with semantic aspects of layouts, such as naming or grouping of elements. These decisions are currently left to the designer, but could be incorporated in future versions if models for these aspects are found.

2.5 Dynamic Layout Optimisation during Sketching

Layout design is a high-dimensional, NP-hard problem [35, 136]. Search is computationally expensive, as multiple objective values are calculated. Yet, when integrated with fast-paced sketching, only brief computation times are allowed. Further, the design problem is also under-specified, because the designer may not have time to specify point objectives and assumptions for each sketch.

Our approach builds on three ideas: First, we relax the requirement to specify a design task to an optimiser. Instead, we infer the "implicit" design task assumed in the sketch currently edited. The internal representation of the *design task*, in its simplest form, is comprised of the different layout elements placed on the canvas, and their 'importance' values, if specified. Using this, We can search alternatives without asking the designer anything. Second, we relax the assumption that the goal is to find the optimum design. Instead, we run several optimisers simultaneously that *explore* the design space with different assumptions. We pick a few diverse design ideas and pass them to the designer, and let his/her choices guide the search. Third, to accelerate search, we make use of the observation that there are far fewer (abstract) *design tasks* than there are (concrete) layouts. Using this observation, we pre-train an "associative memory" that has a good starting point for many design tasks that might occur. When Sketchplorer is running, it maps the present sketch to the closest seed in the associative memory to offer a good solution very quickly. By contrast, MenuOptimizer assumed a point-optimisation goal and en-

forced a complete reset of the optimiser's state when a single element was changed [9].

2.5.1 Definition: Layout Design Task

Layout design is similar to the letter assignment problem from keyboard optimisation [21] and to the *layout facility problem* (LFP), where the task is to place machines with fixed/varying size in a workshop [35, 136]. Layout design in HCI, as we address it here, is a special case, where the goal is to find a non-overlapping planar orthogonal arrangement of n coloured rectangular elements so as to minimise the cost function (see below). There are two differences to LFP: 1) elements can be coloured; and 2) the cost function is more complex. We make two further additions to the task. First, we define the task to involve *element types*, each with unique size constraints. Second, we allow groups. The elements can be nested as long as their bounding box edges do not overlap.

Internally, our optimiser represents a layout as a 2D array of pointers to information objects which store the colours, usage probabilities, etc. of the elements. We use an efficient maximal empty rectangles algorithm [150] to find places in this array where we can move or grow elements to modify the layout. Additionally, each element also has an internal array representing the positions of any child elements relative to itself. We treat the canvas in a recursive loop from the highest to the lowest level of this hierarchy.

2.5.2 Objective Function

We define a multi-objective task where we seek to minimise a weighted combination of the outputs of the five models:

$$U = \sum_{i=1}^{5} w_i S_i, \tag{2.6}$$

where the weights w_i sum to 1 and the individual objectives S_i are normalised to the range $[0, 1]$.

In addition, we introduce two optional canvas-level constraints. We compute *alignment* following an approach similar to [11], in which we count the number of 'alignment lines', or object edge positions, weighted by the average size of object edges lying on those lines. We compute *packing efficiency* as the proportion of the minimum bounding rectangle for the layout elements which is empty, averaged with the proportion of the overall canvas which is empty. These help the optimiser improve quicker in the grid quality dimension. They align elements horizontally and vertically and "pack" them. However, since they aggressively push search toward harmonic layouts, it may drive search to a local minima. We address this by introducing a filtering stage where we compare results for solution quality and diversity (see below).

2.5.3 Inferring the Design Task

We extract the *design task* by analysing the present layout being edited by the designer. We map a layout to a design task by 1) counting the

Sketchplorer Feature	Duration (s)	Search method	Neighbourhood size and type	Objective function
Recolour	1-10 s	VNS	Small; only colour changes	Colour only
Fix	1-10 s	VNS	Small; only spatial changes	All
Explore	1-10 s	AM	Large; only spatial changes	All
Explore	10-120 s	VNS	Medium; only spatial changes	All

VNS: Variable Neighbourhood Search, AM: Associative Memory

Table 2.1: Sketchplorer uses a multi-threaded optimisation strategy that both explores and exploits. To support dynamic changes in the task, two methods are used in parallel: Variable Neighbourhood Search and Associative Memory.

number of occurrences for each element type, and 2) making a distinction between high and low importance items for each element type.

2.5.4 Dynamic Optimisation

During sketching, the optimiser system runs several optimisers simultaneously, each exploring the design space differently, and dedicated to a functionality in the Sketchplorer workspace. An overview is given in Table 2.1. The optimisers deploy two search heuristics.

Variable Neighbourhood Search (VNS) [54] is a strategy combining a neighbourhood-based meta-heuristic to a hill-climbing algorithm. This strategy was chosen to ensure good exploration of the design space, since it is able to increase the search radius if the search gets

stuck. It allows larger and larger changes to be made in each iteration. In our case, we run a steepest descent with different pre-set neighbourhood sizes. A neighbourhood is defined as a change to the colour, position, or size of an element. Small neighbourhood means that one iteration can only produce one change to a layout. Large neighbourhood means that multiple changes are allowed. The optimisers use VNS with small (e.g. $1, 2, 3$) or medium radii (e.g. 4 to 10). Figure 2.4 shows good and failed designs produced in about 4 minutes on a laptop computer. Since Sketchplorer is targeted towards skilled designers, we take a mixed-initiative approach [63], and rely on the designer to identify good designs, and filter out unappealing or illogical ones.

Associative Memory (AM) [163] is executed in parallel. AM is a memory-based learning approach to dynamic optimisation problems. The idea is to populate the memory with good solutions offline, and map the current design task to its closest-matching task in AM, and use that as a starting point for search. We implemented *layout remapping* by searching AM for layouts with design tasks that were supersets of the current design, ordering the elements in each by usage probability, then searching for matched type pairs. We trained our AM for 6,600 randomly generated design tasks. The live optimiser loaded the AM in advance in order to respond quickly (within few seconds).

2.5.5 Filtering and Diversification of Results

Since the parallel searchers explore different parts of the design space, and we cannot show all results to the designer, we pool the results and filter them using a bi-objective pareto front criterion. We define

Figure 2.4: Examples of (a) successful and (b) unsuccessful global designs from the Associative Memory, *before* real-time optimisation. Designs were computed from randomised starting points, in 4 minutes each.

this as the weighted (and normalised) sum of the objective score and *inter-layout distance*. Inter-layout distance is a score from 0 to 1 where 0 means identical layouts and 1 means that all elements have different locations. This diversifies search results and prevents identical designs from being shown to the designer.

2.6 System Implementation

The Sketchplorer tool is implemented in Objective-C. Multitouch input is detected using PQ Labs multitouch SDK, and interactions are recognised using customised gesture recognition. The optimisers are written in Python, and use parallelisation for speed-ups. Given no prior information, the optimiser assigns equal weights to each of the five models ($= 0.2$). This could, however, be tuned to further steer the optimisation. When designers make changes to the current design, layout files (in JSON format) are sent from the design tool to the optimisers asynchronously. The optimisers run on dedicated laptops (we use Macbook Pros), and all devices communicate using ThoMoNetworking. Local vs. global design changes are identified using filename tags. To avoid outdated retrievals, the optimisers kill existing threads when a new file is received. Generated designs are immediately pushed back to the design tool.

2.7 Study 1: End-User Evaluation

Our first study addresses optimisation of the design of the Windows Phone home screen. This study involves no designer in the loop. The goal is to evaluate our optimisation approach by testing its outputs with *end-users*. As the use-case scenario, we chose the Windows Phone home screen, a complex design task that could not have been addressed with previous approaches. This task allows the inclusion of several interactive elements, with multiple icon sizes, colours, and positions, spanning over three pages. We compare an optimised de-

sign against the baseline in a study where users are asked to select applications from the home screen. As our baseline, we implemented the factory default of a Lumia 930 phone. Apps which were not pre-installed on the phone were added in a random order at the end of the default menu, as in an unpersonalised order-of-installation layout. Comparing against a commercial design provides us with a realistic baseline for the layout optimiser.

2.7.1 Optimisation Task

The inputs to the optimiser are a list of 55 apps, together with usage probabilities from the LiveLab database [134]. We sampled the primary colours from the app icons manually. When generating layouts in the optimiser, we allowed as many 8×6 grid pages as necessary, and each icon was sized at 1×1, 2×2, or 2×4 grid cells. These values match the standard grid and icon sizes of Windows Phone home screen layouts, which is used as the baseline. For some apps, the icon is coloured according to system-wide accent colours. Others have icons with developer-chosen colours. For the system-coloured icons, the optimiser was free to choose from a set of 20 perceptually distinct colours [74].

Offline optimisation follows a different approach than in the real-time, dynamic optimisation task. Here, we performed multiple restarts of a random search procedure with different objective weights. For each weight set, we generated and evaluated 10,000 random layouts, then performed 3000 iterations of local search around each of the best three designs.

The menu with the best score over all weight sets was chosen as the design to evaluate with users. The models predicted that our optimised design would be better than the baseline in terms of visual search (score 0.106 vs. 0.117, a difference of ∼270ms), selection (0.108 vs 0.104, ∼90ms), and grid quality (0.005 vs. 0.013), similar in colour harmony (both 0.015), and worse in terms of clutter (0.028 vs 0.020).

2.7.2 Participants

We recruited 20 student participants (4 male), aged 20 to 36 (mean 26.7, s.d. 5.2). They had 2 to 8 years of experience with smartphones (mean 4.3, s.d. 1.8). Participants were required to not have prior experience with Windows Phone devices. Two participants were left handed. Participants were compensated with a cinema ticket.

2.7.3 Apparatus, Procedure, and Experimental Design

We implemented a logging application to display the menus and collect performance information on a Nexus 5 smartphone running Android 4.4.3.

Participants performed 275 trials with each menu. In each trial, they were shown a textual stimulus with the target name. After tapping to dismiss the stimulus, the first page of the current menu was shown. Participants had to navigate to and tap the appropriate icon. A hint button was available at all times, to remind participants of the stimulus. The timestamps for all taps and swipes were logged, along with any incorrect selections. As stimulus, different apps were shown several times. The exact number of times each app was displayed

ranged between 1 and 25, depending on their usage probability. We used a smoothed version of the distribution used to train the optimiser, in order to increase the number of applications with more than 1 selection.

Participants performed the experiment while seated in a quiet room. They were asked to hold the phone in their non-dominant hand and tap with the index finger of their dominant hand. The order of conditions was counterbalanced.

After this, participants were shown images of the full layouts and asked to rate them from 1 to 5 in terms of overall aesthetic quality, use of colour, composition logic, and symmetry. They were also asked to indicate a preferred layout.

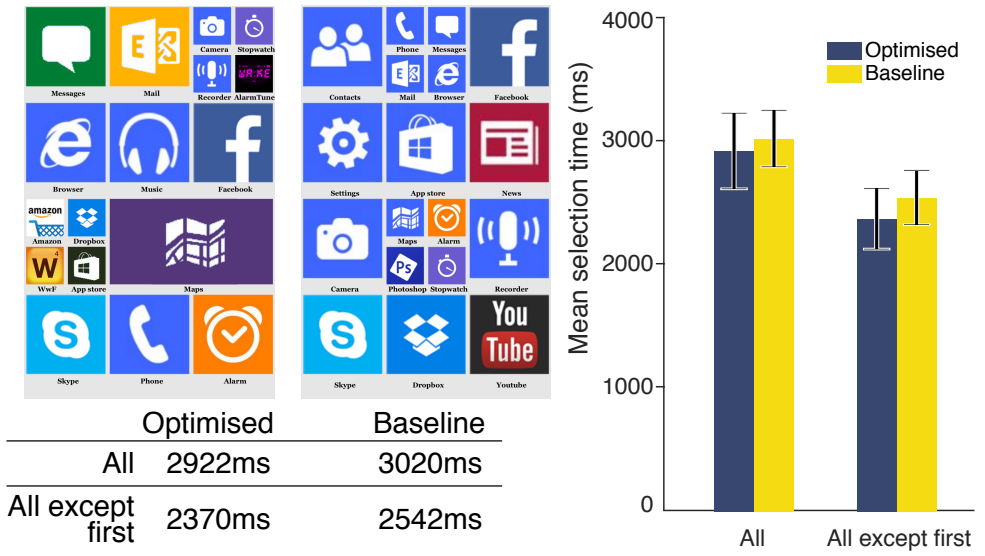
2.7.4 Results

Selection time and measures of perceived aesthetics are the dependent variables used in this study.

Selection Time

As our principal performance metric, we use the *average selection time* over all trials, measured from the tap to dismiss the stimulus to the tap on the correct icon. We filtered out trials with selection errors or hint requests.

Figure 2.5 shows our results and the first page of each of the designs. Over all trials, there is no significant difference between the average selection times (optimised: 2922ms, baseline: 3020ms) between the two menus (paired t-test, $t(19) = -0.95, p > 0.05$).



	Optimised	Baseline
All	2922ms	3020ms
All except first	2370ms	2542ms

Figure 2.5: *End-User Study.* Left: Homescreens of the optimised and baseline designs, and corresponding average selection times. Right: Graphs for average selection times (95% CIs). The optimised design was significantly faster when the first selection for each app was excluded.

However, when the first selection for each app in each condition is removed from consideration, the difference in selection times (optimised: 2370ms, baseline: 2542ms) between the menus is significant (paired t-test, $t(19) = -2.67, p < 0.05$). This removal is justified given that users were mostly familiarising with the menus during the first selections. We also analysed only the latter half of trials for each app, to see if the learning benefit continued to grow. However, the results when doing this were very similar. This suggests that the majority of the learning takes place on the first selection. This result matches with our models that predict expert performance rather than novice.

Aesthetic Ratings

For the aesthetic ratings, we found no significant differences between the overall ratings and logic of composition scores across the designs (Wilcoxon signed rank test, $Z = -1.80, -1.79, p > 0.05$). However, users rated the use of colour in the optimised design significantly higher than the baseline ($Z = 2.38, p < 0.05$), but the level of symmetry significantly lower ($Z = -2.65, p < 0.05$).

Half (10) of the 20 users expressed a preference for the optimised design over the baseline. Several participants noted that they disliked one or both designs because the menus did not prioritise the apps they themselves used. This reflects the app usage dataset, which is several years old.

2.7.5 Summary

To summarise the results from the user study, the optimiser was able to produce some performance benefits over a realistic baseline design. The observed differences were largely predicted by our models.

2.8 Study 2: Design Study with a Live System

To evaluate *Sketchplorer* as an integrated sketching and exploration tool, we conducted a user study with experienced designers. We aimed to evaluate whether Sketchplorer is effective in supporting creativity and problem-solving in sketching. We were interested in designers' insights and impressions, and not on validating the produced designs themselves, as this had been investigated in the previous

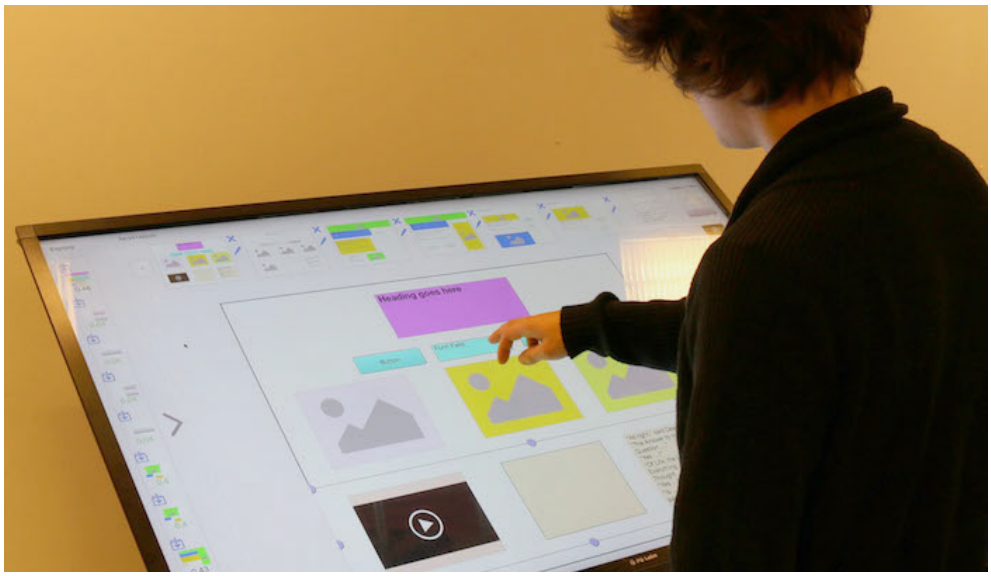


Figure 2.6: *Designer Study.* We evaluated Sketchplorer with 10 trained designers. Participants completed design tasks while using the different features supported by the design tool.

study. During the study, participants were given design tasks, and asked to use the tool to create viable solutions. A free-form, and open-ended, methodology was used during the study. We did not direct or enforce participants to use the optimiser's suggestions but were interested in seeing if this would happen spontaneously and organically.

2.8.1 Study Design

We recruited a total of 10 participants (2 female), age ranging from 22 to 40 years (mean 29.4). All participants, except one, had an educational or professional background in design, and all of them had some experience using digital design tools. Participants were compensated

with two cinema tickets.

All design tasks were performed on a 55-inch (140 cm) 4K display (3840 × 2160), with a PQ Labs G5S multitouch overlay. The display was tilted to a comfortable angle, and participants performed tasks in standing position (Figure 2.6). The design tool ran on a Macbook Pro (OS X 10.10). In addition, two Macbook Pros were used for the optimisers.

Procedure: After initial training, in which participants created simple designs and explored the tool, they were given a 'design brief' for the main study task. Similar to the example in the walkthrough, they were asked to create designs for a blog page. There were no strict requirements, and participants could freely decide on the exact elements, and their details. They were asked to create a few different designs, and could choose to make as many as they wished to, in a time frame of 30 minutes. Participants were free to save designs that were sketched with or without the aid of the optimiser. At the end of the task, they assigned ratings (1–5) to their saved designs, using a custom 'viewer' application. Importantly, the viewer did not reveal whether a given design had been entirely sketched by the participant, or if it was optimiser-assisted. Finally, we gathered further information through a questionnaire and a semi-structured interview.

2.8.2 Results

The study aimed at gauging whether Sketchplorer allowed designers to sketch freely, and their usage of different features the tool provided. On reviewing participants' list of saved designs, we found that 8 out of

ID	N(Saved)	N(Recolour)	N(Fix)	N(Global)
1	3	0	0	3
2	5	0	0	2
3	5	3	1	0
4	5	0	2	2
5	4	1	0	1
6	3	0	0	0
7	4	0	3	0
8	8	0	0	5
9	10	2	3	4
10	1	0	0	1

N(Saved) : Total number of saved layouts
N(Recolour) : Saved layouts with recolour-suggestions
N(Fix) : Saved layouts with fix-suggestions
N(Global) : Saved layouts with global-suggestions

Table 2.2: Summary of features used by each participant in Study 2. 9 out of 10 designers took advantage of suggestions offered by the optimiser, to achieve their final designs.

10 participants had at least one optimiser-aided design in their saved list. Additionally, participant 10 stated to have borrowed an idea from the optimiser suggestions, but recreated it manually in his own sketch. Optimiser-assisted designs received an average rating of 3.47 out of 5 (S.D: 0.91), and unassisted sketches had an average rating of 3.48 out of 5 (S.D: 0.77). Thus, designers were equally satisfied with optimiser-generated designs as they were with their own work.

Table 2.2 summarises the optimiser-features used by the participants, in the saved versions.

Responses to the questionnaire and interview session were encour-

aging for Sketchplorer. Apart from minor technical glitches, participants commented that they could sketch out their ideas quickly and freely, and the multitouch interactions were easy and straightforward. The ability to have a large touch-friendly canvas was appreciated. All participants stated that they would certainly prefer a visual timeline in such design tools, over traditional save-dialogs, and such a feature would be useful in future tools.

With regards to the optimiser-related features, participants especially liked the *explore* option. They found the suggestions to be distinctly different, useful, and indicated that it would help them while designing. One commented, *"I kind of like how the suggestions turn my approach upside down, and I can get hints from there"*. There was a split between participants while rating the usefulness of *fix* and *recolour* during the given design task. A few participants found some of the *recolour* suggestions *"so 90s"* or *flashy*, and wished that the suggestions would be more subtle. One participant commented that he would rather explore new and different ideas, than 'fix' existing ones. However, participants also indicated that recolouring and fixing would help them in future design activities.

2.9 Discussion

2.9.1 Summary

This chapter has studied a new concept for interactive design optimisation, backed by predictive models, and integrated within a sketching tool. Sketchploration enables designers to improve placement of inter-

active elements on a GUI, at design-time. By inferring design tasks implicitly, and by using a combination of exploration and exploitation, a layout optimiser can complement the sketching activities of a designer in real time, and facilitate them to explore larger design spaces without having to divert their attention to the optimisation process. During sketchploration, the designer–optimiser system is continuously both sketching (or, in optimiser terms, exploiting) and exploring. Crucially, the optimiser is not suggesting just anything for the designer. The predictive models of the optimiser try to "pull" the designer towards usable and aesthetic designs. A designer can reject designs that are unsatisfactory for some reasons that the optimiser may not include in its objective function. This way, the designer and the optimiser can iteratively approach a region of good designs without communicating an objective function. We are not aware of a similar approach in the past.

To verify our approach, we conducted two user studies. Study 1 exposed end-users to a visual layout generated by the optimiser, comparing it against a commercial baseline design. The results were favourable, the optimised design was significantly better in average selection time, when measured after the first selection of an app. Colour harmony was rated higher, but the layout less harmonic. It is intriguing to note that the obtained data are mostly in alignment with the predictions made by the models. Study 2 gauged Sketchplorer's ability to provide designers with a sketching environment conducive to sketchploration, and gathered insights about the different features. Most designers in our study added some of the optimisers' designs to their saved list of designs, and used the optional features in their

design process. Among the three suggestion types available (fix, recolour, and explore), designers particularly appreciated the explore feature. While scoring the final designs they had created and saved, on an average, they rated optimiser-assisted designs to be as good as their sketched designs (mean rating = 3.5 on a scale of 1 to 5). Given that the participants had educational and even professional background in design, we consider this a notable achievement.

Overall, sketchploration can be seen as a promising concept; one that can lead to not only empowering designers, but also to systematically improving usability and aesthetics, and raising the bar of design.

2.9.2 Revisiting the Research Question

The goal of this chapter was to investigate concepts and methods to construct a graphical interface at design-time such that designers could improve the placement of interactive elements on the layout. The research question, as stated in the introduction, is:

*How can we computationally support designers in the process of design **exploration** during early stages of placement of interactive elements on a graphical interface?*

The work presented in this chapter has addressed this research question by making the following technical contributions:

1. *Design principles* for integrating a model-based interface optimiser to a sketching tool.
2. *Extension of model-based interface optimisation* to interactive layouts by: 1) Formulation of (unconstrained) layout sketching as an optimisation problem, allowing for addressing the positions, sizes,

and colours of elements. 2) A novel, theoretically informed objective function addressing five aspects important in layout use: perception of clutter, visual search performance, pointing performance, grid quality, and colour harmony.
3. *A dynamic optimisation approach* that 1) infers the designer's task from the current layout, 2) simultaneously explores and exploits in the design space and 3) reacts rapidly to changes in the current sketch.
4. *The Sketchplorer design tool* that integrates this approach into a sketching and design exploration tool.

Quantitative and qualitative results from two studies provide first evidence for the concept and our approach.

2.9.3 Principles for Design-Time Placement

The key design principles derived from this work, towards the goal of improving placement on GUIs at design-time are:

1. **Support quick and ambiguous placement**, and leave room for uncertainty, by providing capabilities to defer the task of specifying details at any stage of the design process.
2. **Allow fluent shift in focus** between exploration and refinement.
3. **Support for multiple hypotheses**, and give an overview of progress, by providing a non-destructive, and editable, timeline of previous alternate designs.
4. **Minimise user input** not related to the placement activity by inferring details whenever possible.

5. **Eliminate ambiguity** in designers' minds by communicating with the designer in a timely and predictable manner.

2.9.4 Limitations and Next Steps

While our work lays the foundation for sketchploration, it also uncovers many challenges and opens up opportunities for future efforts. First, while this chapter covers visual aspects of designs, it does not capture semantics or the dynamic aspect of interactions that interfaces afford. Additionally, it is restricted to elements with rectangular footprints. Relationships between elements, such as logical sequences, relative importance, and adjacency play a critical role in the design [132]. To further improve exploratory results, integrating these aspects in the future is desirable. Integrating more shapes and widget types will also increase the impact of such a system. Second, our optimisation techniques are highly dependent on well-validated models; further development of accurate predictive models is a key factor to improvements in results. Third, although the optimiser's results mostly had high scores, we note that its performance was limited to designs with ten or fewer elements. Layout optimisation for HCI is a combinatorially challenging task that cannot be directly solved with standard methods. Finally, it can be beneficial to allow deeper exploration into design spaces, and provide designers with detailed insights of the entire space. In optimisation terms, the optimisation landscape could be visualised to allow more informed choices. These improvements can provide even better results during design-time placement, and aid designers in creating better graphical interfaces. However, design-time

placement has the general drawback that it caters to *the user*, and not specifically to *a user*. Designs can not be tailored and customised to individuals, and remain static once implemented. To address this limitation of design-time improvements, in the next chapter, I investigate the potential for *use-time* improvements to placement of elements in graphical interfaces, by dynamically adapting designs to individual users.

2.10 Acknowledgements

The research was conducted while I was a summer intern at the User Interfaces Group, at Aalto University. The project received funding from the Academy of Finland project COMPUTED and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 637991).

Apart from my co-authors, Daryl Weir and Antti Oulasvirta, I would also like to thank Andreas Karrenbauer and Gilles Bailly for their suggestions and comments, Olli Savisaari and Perttu Lähteenlahti for their assistance, and all study participants for their time and engagement.

Chapter 3

Improving Use-Time Placement on Graphical Layouts

The previous chapter addressed design-time challenges in improving the placement of elements on a graphical interface layout. It improved the design exploration process for interface designers during sketching activities. In this chapter, I discuss user-sided challenges, and present a strategy for placing elements, such that interface layouts can adapt themselves to individual users after they have been deployed. In contrast to *real-time* adaptation, where an interface adapts itself while being used, *use-time* (or just-in-time) adaptations are executed when the interface is visited, immediately before it has been presented.

In domains where users are exposed to large variations in visuospatial features among designs, they often spend excess time searching for common elements (*features*) in familiar locations. This chapter contributes computational approaches to restructuring layouts, and plac-

ing features on a previously unvisited interface such that they can be found quicker. To achieve this, four concepts of *familiarisation* are explored, inspired by the human visual system (HVS), to automatically generate a familiar template for each user. Given a history of previously visited interfaces, we restructure the spatial layout of the new (unseen) interface with the goal of making its elements more easily found. *Familiariser* is a browser-based implementation that automatically restructures webpage layouts based on the visual history of the user, captured from users' browsing activities. An evaluation of the system with users provides first evidence favouring familiarisation.

The contents of this chapter are based on the following publication:

1. **KASHYAP TODI**, JUSSI JOKINEN, KRIS LUYTEN, and ANTTI OULASVIRTA. Familiarisation: Restructuring layouts with visual learning models. In *Proceedings of the 2018 ACM Conference on Interactive User Interfaces*, IUI '18. ACM, New York, NY, USA, 2018

3.1 Introduction

This chapter addresses a common predicament in interaction with graphical user interfaces— from blogs to banking and mobile apps— users encounter a wide diversity of visual designs. Even when serving the same purpose, designs differ in terms of element positions, colouring, images, and widget types. While visual uniqueness provides for identity and brand recognition, it also implies that users are constantly confronted with learning, relearning, and accustoming to navigating new structures and different styles. For the visual system,

this poses a challenge to constantly adapt the use of visual attention. By understanding how this happens, we could design and adapt interfaces automatically such that elements can be more easily found. This could make interface ecologies more usable and enjoyable for users who care less about brand identity.

3.1.1 Research Question

Our work is motivated by the observation that the added effort experienced upon encountering unfamiliar or atypical designs depends on the cost related to visual search for new and unfamiliar visual layouts [28, 69, 133]. Consider taking a familiar design and moving an element to a different position. Users would first seek the element in its expected place and, upon failing, continue with some search strategy to locate the element that was moved, or simply give up. However, designs are likely to differ in more than one feature, requiring 'guessing' of element positions, adding further frustration.

The potential benefits of placing interactive elements on a GUI at use-time, based on a user's familiarity, inspire the following research question:

How can we adapt graphical interfaces for individual users' by automatically placing elements at familiar locations, at use-time, such that they are consistent with a user's mental model and enable faster visual recall?

To address this research question, this chapter investigates the concept of *familiarisation*, and presents computational principles for restructuring unfamiliar designs to designs the user considers to be familiar.

3.1.2 Familiarisation: Restructuring Graphical Interfaces using Visual Learning Models

We define *familiarisation* as a technique to adapt and restructure a new, unvisited interface layout based on what a user has previous seen and learnt. Formally, the goal of familiarisation can be defined as follows: *Given a new design d, unfamiliar to the user, and a history of previously visited visual designs H ($d \notin H$), restructure d to minimize costs to visual search to the user.* By 'restructuring' we refer to manipulations to the visio-spatial layout, such as moving, resizing, and recolouring elements. We call techniques that achieve this *familiarisation techniques*. These techniques exploit the capability of operating systems and browsers to change an interface design dynamically. Such computer-driven familiarisation complements efforts at design-time to ensure *consistency* and adherence to design standards and guidelines [105, 123]. At the user-end, familiarisation exploits the known processes of human visual system in visual search. This ensures that the familiarised layouts are more usable due to predictable element locations and layout structures. While consistency enforced at design-time can only approximate what the user population has experienced, ensuring consistency through active familiarisation at use-time allows, in principle, 'perfect consistency' for an individual.

There are multiple competing principles on which familiarization can be based, depending on how familiarity is defined. As these principles can lead to different techniques of familiarisation, it is important to explore their assumptions, implementations, and results. To this end, we define and formalise four principles of familiarisation, in-

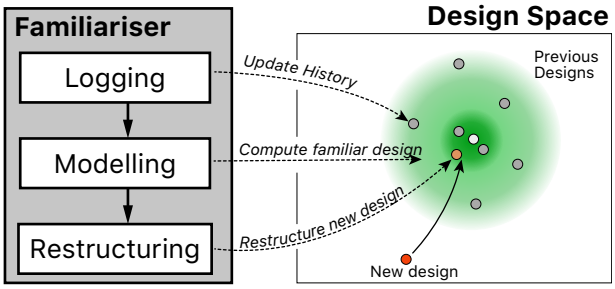


Figure 3.1: The three stages for familiarisation. First, user's data is logged and stored as history. Each time a page is visited, the familiarity models are updated, and the base design is computed. The new page is then adapted to match the base design.

formed by theories of human visual system (HVS) and visual learning. These are implemented in *Familiariser*, a browser-based system that dynamically restructures interface layouts, at use-time, to make them easier to use. Figure 2.1 illustrates the conceptual design of Familiariser, consisting of three main components: 1. User history logging; 2. Modelling familiarity; 3. Restructuring a new page. Results from our study indicate that familiarisation can reduce movement times by over 10%, and also leads to over 20% lesser eye-gaze fixations.

3.1.3 Overview: Four Familiarisation Principles

Familiarisation techniques can be generally defined as functions that take as input user history H and a new design d, and yield a restructured design d': $f(d, H) \rightarrow d'$. In this chapter, we explore four principles, illustrated in Figure 3.2, with different theoretical underpinnings and consequences to how restructuring operates and how they influ-

ence results. Section *'Modelling Familiarity'* elaborates on each of these principles in greater detail.

I. Frequency: The most straightforward approach is to simply take the most *frequently used design* from H. Any new design within the same domain would be restructured such that elements appear in roughly the same places. While this design is likely to be recognizable, and simple to implement, the approach does not take into account the fact that most recent experiences are likely to dominate recall. Something that might have been frequently used in the past might be partially forgotten and overshadowed by the most recently encountered designs.

II. Serial Position Curve: The second principle implements the well-known serial position function of long-term recall [42]. Extensive empirical research on long-term and short-term memory has shown that the first and the most recently encountered objects are better recalled than ones in the middle. We implement a mathematical function that describes the relationship between order and recall probability. This is used to choose the most-likely-to-be-recalled design in H according to this function. A limitation of this and the first approach is that the visual features might not be adequately described by a single design.

III. Visual Statistical Learning: The third principle is visual statistical learning (VSL), according to which visual attention relies on a probabilistic internal model sensitive to the characteristics of the environments they have encountered before. We implement this hypothesis by building a statistical model of visual features in H. It is used to estimate the 'most probable' location of a given element. We then re-

structure a layout, element by element, considering this function, and finally align the elements so that they appear orderly.

IV. Cognitive Model: In our final technique, we train a cognitive model of layout learning with H and use its prediction on an empty canvas to predict the most likely positions of the elements of d. The model simulates learning of visual positions, and generates visual search patterns and times, given a layout and knowledge on that layout [69]. As an input it requires H and associated visitation durations, and optionally also considers element frequencies or relevance of elements. In this, it is similar to the previous approach. However, the model generates the actual eye movement patterns and visual search times, in addition to generating the memorability of location elements. It also simulates forgetting of layouts that have been visited only shortly and further back in time. This allows the model to be used for evaluating automatically how the familiarised layout will impact the user's behaviour, as well as how the user learns the new layout.

3.2 Background

At its core, our work is situated within the domains of (1) visual search modelling, (2) layout generation and restructuring, and (3) user interface adaptation. We discuss prior literature that has dealt with these themes, and draw comparisons to our work.

3.2.1 Visual Search

Visual search requires the user to scan through a *search array* to find a target [122]. This process is shaped by the limitations of the human vision and information processing. The area of clear visual acuity (*fovea* is limited, and therefore in the case of UIs, the user is able to see only partially its graphical elements at any given time. In visual search, the human visual system is therefore faced with an attention deployment problem: what visual element to attend next?

Attention deployment during visual search has been modelled using various computational models [78]. A popular computational search model computes salience of regions in an image, and uses that to predict what the most probably attended regions will be [66]. In also implements an *inhibition of return*, where already searched regions are not revisited. This search model can be called *bottom up* model, because its attention deployment computation is based on features of the visual field. Another approach, *top down*, emphasises the role of the search task. In the case of UIs, the user probably has some task-relevant information, which can be used to guide the search. A model of visual search on keyboard layouts utilised this kind of approach, modelling the impact of learning on visual search [69].

3.2.2 Layout Generation and Interface Restructuring

Automatic generation of layouts, and restructuring existing designs, have received significant attention by researchers and practitioners. One of the main rationales behind automatic generation is that it simplifies or eliminates the design task, making it possible for program-

mers or non-designers to create interfaces. Restructuring of designs allow for systematic improvement in usability and aesthetics of interfaces. Prior works have often used rules and heuristics to generate layouts that adhere to specific design guidelines [7, 19, 107, 108, 115], or example-based retargeting by mining existing designs [81, 82]. Our work situates itself in the area of model-based interface generation. Model-based techniques [16, 19, 115, 132] tackle the design problem by first abstracting the user interface as a set of models, which are then used to steer the generation or restructuring. There have been several prior systems, dating back to the 1980s, such as [9, 67, 103, 109, 146], that apply model-based approaches to interface (re-)design.

The above techniques and implementations have largely considered the user population as a whole, and not individuals or groups of users. There has also been some significant work done to address different groups of users. Ability-based design [159] focuses on creating interfaces that take into account different requirements or restrictions, and optimise them towards specific abilities (e.g. [46, 127]). Culture-based design [75] took into account differences among different cultures, and used this as a basis for creating multiple designs of the same interface, each suited to a specific group of users. While all these works have considered a subset of the population, or specific preferences and requirements of groups of users, they generally do not address design on the per-user level, and also do not take into account individual users' past experiences or usage history. In contrast, with SUPPLE, Gajos et al. [43, 45] defined user interface generation as an optimisation problem that took a set of user interactions as input, and used this to generate an optimal solution. By doing so, the system

could take into account specific user requirements and abilities, and create individualised interfaces for each user. HIGHLIGHT [102] enabled re-authoring of existing websites based on tracing how the user interacts with the site, and created mobile versions customised to the user tasks and mobile devices. UNIFORM [104] bears a lot of resemblance to our work, as it takes into account a user's history to automatically generated remote control interfaces. UNIFORM differs from our work since it does not restructure existing interfaces, yet focuses on generating consistent user interfaces that provide access to existing functionalities on new platforms. Additionally, the user history is limited to one source design, and the work focuses on the engineering aspects of restructuring an interface to match the source, and not on the systematic selection or generation of source designs from an extended history. The general approach adopted by previous works on per-user model-based interface generation has been to model the user as a one-time activity, before generating an optimised interface. Familiarisation, on the other hand, captures the complete user history, and each time a new interface is encountered, it generates a just-in-time redesigned interface based on an updated familiarity model. Interfaces can thus evolve over time and usage, and (re-)design becomes a continuous activity.

3.2.3 Run-time Adaptation of Interfaces

As highlighted above, familiarisation presents an approach to adapt and restructure interfaces at runtime. Such an approach falls under the category of 'adaptive user interfaces'—interfaces that can adapt or

modify themselves while being used. There have been several works on adaptive UIs previously. Past systems have used different criteria for adapting interfaces. For example, the Walking User Interface in [159] adapted based on whether the user is stationary or walking. In the [44], Gajos presents UIs that adapt to users' current tasks. In contrast to use scenario or current task, Familiariser adapts an interface layout based on users' history, and exposure to previous interfaces belonging to the same domain. A common criticism and shortcoming of adaptive UIs is that unpredictability and cost of adaptation can negate the benefits provided [49, 84]. Since we focus on adapting newly visited interfaces that the user has not previously used or learnt, it does not suffer from these drawbacks. Thus, by incorporating model-based design generation, and just-in-time interface adaptation, our work attempts to leverage recall and visual learning, and provide users access to interfaces tailored to their expectations and mental models.

3.3 Modelling Familiarity

What we call 'familiarity' is a complex cognitive phenomenon influenced by several factors related to how people learn visual search. We approach the problem by exploring progressively sophisticated principles inspired by theories of HVS and human memory.

At the highest level, these principles break down to two groups: (a) page-wise familiarity; (b) feature-wise familiarity. In *page-wise familiarity*, an entire layout (e.g., webpage) is recognised as one unit. It is assumed that users learn and recall entire pages and their visual layouts. On the other hand, feature-wise familiarity considers high-level

features on webpages as individual units. For example, logos, navigation menus, and search-boxes, can be considered as features, and users learn and recall these recurring features across different pages. Based on these two cases, we explore four principles:

3.3.1 Principle I: Frequency

This is the most straightforward *page-wise* approach to defining familiarity. It is informed by the frequency effect: frequently encountered items are more likely to be recalled than less frequently encountered. More specifically, the number of encounters directly affects both retrieval time and retention probability: more frequently practised items are recalled faster and easier [5]. The interface (webpage) that has been encountered the most number of times (i.e., most frequently visited), and for the longest durations, is assumed to be the most familiar to the user, and is thus used as the base design to which new pages are familiarised:

$$f_{page} = n_{visits} * t_{average} \tag{3.1}$$

where,
f_{page} = familiarity score of a page;
n_{visits} = number of visits;
$t_{average}$ = average duration of visits (in seconds).

The result of frequency-based familiarisation in a one-to-one retargeting of the new page to the most-frequently visited page.

3.3.2 Principle II: Serial Position Curve

While frequency of usage can be a reasonable method to predict the most familiar interface in simple scenarios, as user histories get larger over time, this is susceptible to failure. Usage-based aspects such as first exposure to a page, recency of visits, and intervals between visits, are not considered by the first model. Firstly, not only frequency of rehearsals, but also their recency, affects recall time and probability: recently encountered and practised items are recalled faster and more probably [5]. In addition to recency, there is also an effect of primacy, where items that have been encountered first are better remembered than later items [59]. Together, the primacy and recency effects create a U-shaped curve (a *serial position curve*). It maps items encountered in the past to probability of recall: the first and last items are more likely to be recalled better.

This leads to a *page-wise* approach that considers:

1. *Frequency (v):* The frequency of visits to a given page, denoting how often a page is visited.

2. *Recency (r):* The recency of visit to a page, denoting when the page was last visited.

3. *Primacy (p):* The order of visits to different pages, or the sequence in which unique pages are encountered.

For each of these, scores are calculated for every page in the history by ranking them. By aggregating scores for all factors, for each page, we can calculate a "familiarity score" for every page, and hence de-

termine the most familiar interface. Thus, we compute the following scores:

1. *Frequency Score (S_v):* This is the ratio of the visit count for the given page to the total visit count of all pages in the history.

$$S_v = n_{page}/n_{total} \tag{3.2}$$

   where S_v is the frequency score, n_{page} is number of visits to page, and n_{total} is total number of visits to all pages.

2. *Recency Score (S_r):* This takes into account the decaying aspect of memory. The number of other pages that are visited since the user last visited the given page negatively influences the familiarity. The score is normalised (most recently visited page has a score of 1), and this reduces as recency increases

$$S_r = 1 - r_{page}/n_{total} \tag{3.3}$$

   where S_r is the recency score, r_{page} is recency of a page, and n_{total} is total number of visits to all pages.

3. *Primacy Score (S_p):* Pages that are encountered first tend to be better recalled than later pages. This effect is known as primacy, and the *primacy score* takes this into account, where earlier pages have a higher score. The score is normalised (first visited page has a score of 1), and subsequently reduces for following pages.

$$S_p = 1 - ((p_{page} - 1)/n_{pages}) \tag{3.4}$$

where S_p is the primacy score, p_{page} is visit order number for the page, and n_{pages} is number of unique pages visited.

By aggregating these scores, we derive the familiarity score for a page:

$$F_{page} = \alpha * S_v + \beta * S_r + \gamma * S_p \tag{3.5}$$

where F_{page} is the familiarity score for the page, α, β, γ are weights for the three factors, and α + β + γ = 1.

In our implementation, we assign equal weights to frequency and recency, and suggest that order has a lower effect on familiarity, thus assigning it a lower weight. Thus, the values we use are: α = β = 0.4, γ = 0.2.

The page with the highest familiarity score ($max\ F_{page}$) is considered to be the familiar page for a user, and is selected as the basis for familiarisation. The result of serial-position curve-based familiarisation in a one-to-one retargeting of the new page to the selected familiar page.

Figure 3.3 presents a sample scenario, where a user visits four unique pages multiple times. Figure 3.4(a) illustrates evolution in recorded usage metrics and scores, as the user visits different pages, and page selection results for principles I and II.

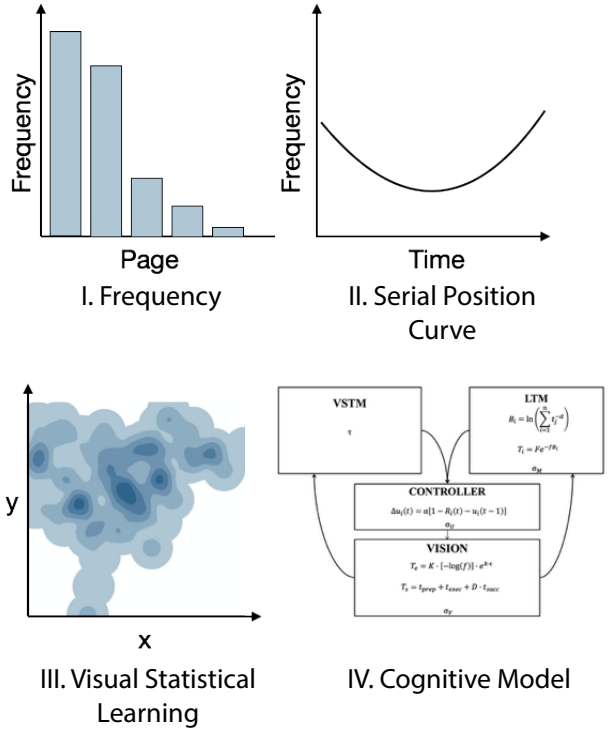


Figure 3.2: An illustration of the four principles for modelling familiarity. (I) relies exclusively on visit frequency; (II) takes time into account; (III) considers positional (xy) information; (IV) uses a cognitive model of visual search.

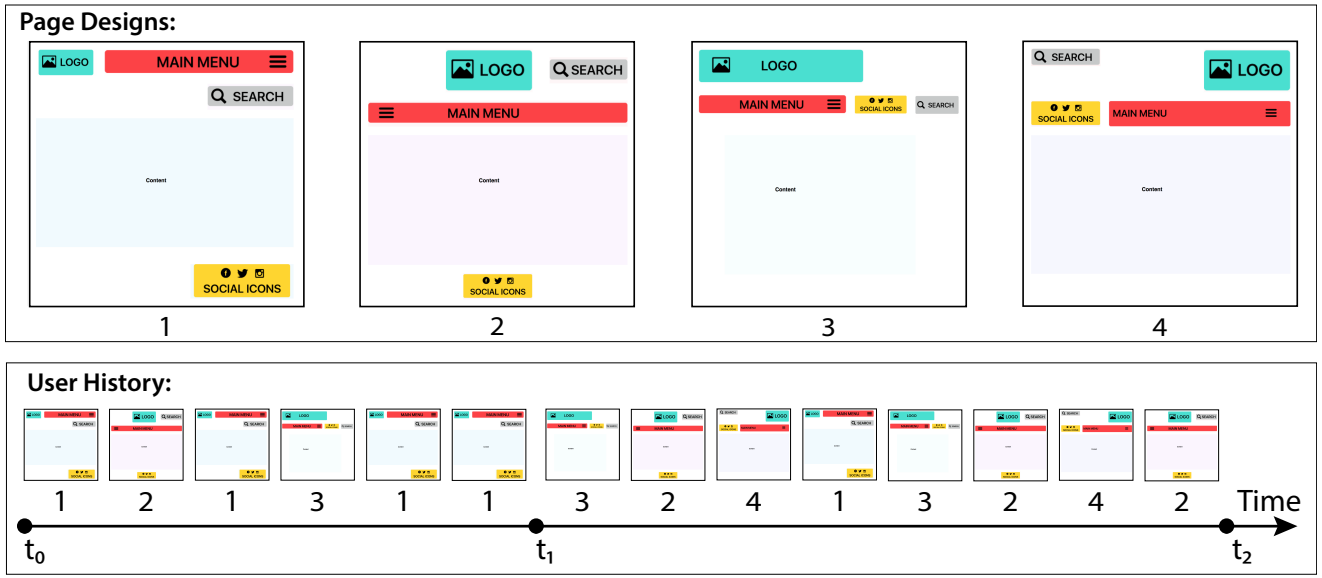
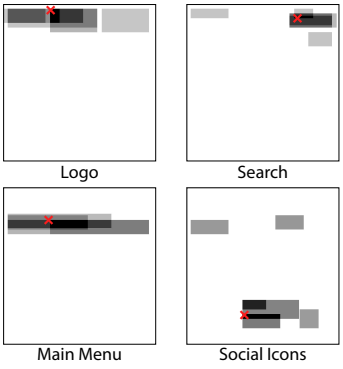


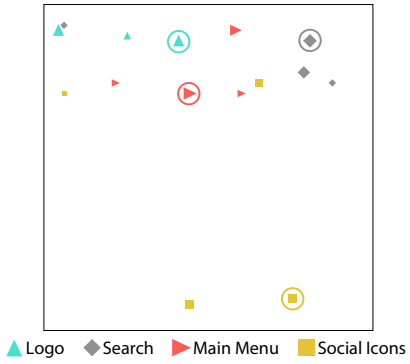
Figure 3.3: A sample scenario with 4 different page designs. The user history shows a timeline of page visits (for simplicity, uniform visit durations are assumed across pages). Model results from applying the familiarity principles at two timestamps (t1 and t2) are illustrated in Figure 3.4.

Timestamp	Page	Frequency (v)	Recency (r)	Primacy (p)	Frequency Score (S_v)	Recency Score (S_r)	Primacy Score (S_p)	Familiarity Score (F_{page})	
t_1	**1**	**3**	0	1	0.667	1.000	1.000	**0.867**	← Selected by I and II
	2	1	4	2	0.167	0.333	0.667	0.333	
	3	1	2	3	0.167	0.666	0.333	0.400	
	4	0	-	-	-	-	-	0	
t_2	**1**	**5**	4	1	0.357	0.714	1.000	**0.629**	← Selected by I
	2	4	0	2	0.286	1.000	0.750	**0.664**	← Selected by II
	3	3	3	3	0.214	0.786	0.500	0.500	
	4	2	2	4	0.143	0.857	0.250	0.479	

(a) Usage Metrics and Scores



(b) Probability Distribution Maps (for III)



(c) Activations (for IV)

Figure 3.4: Results obtained by applying the four different principles to the user history in Figure 3.3. (a) Evolution of metrics and scores at two timestamps (t_1, t_2), and page selection outcomes. (b) Computed probability distribution maps for key features at t_2. Red $\times$ symbols indicate the selected position for each feature. (c) Activations for features at t_2. Circled activations indicate the predicted positions for each feature.

3.3.3 Principle III: Visual Statistical Learning

Here, we take into account the statistical frequency with which different design features occurred in history H. This hypothesis is based on theories of visual statistical learning [25], according to which visual

search strategies are sensitive to the statistical structure of the visual environment.

A *feature* is defined as a high-level semantic element, and can be composed of multiple low-level elements. For example, a website's logo is a feature, and could be composed of an image and a link element. By aggregating the commonly found positions of each feature among the visited pages in a user's history, we can determine the "most familiar" characteristics for each feature, and use this as the familiar template. Unlike in the previous principles, the resultant template design here is not a single page from the history, but a *feature mesh* consisting of familiar features from multiple pages in the user's history.

To generate this feature mesh, or template, we first generate spatial probability distribution maps for each encountered feature. In these maps, for pages in the history, every for every pixel occupied by a particular feature, the spatial probability is incremented. Features are weighed according to familiarity scores (F_{page}) of each page (from #2). Thus, a feature appearing on a more familiar page has a higher influence on the probability distribution. For each feature, the value (position) with the highest probability is selected to create the template consisting of all detected features. Figure 3.4(b) illustrates the computed probability distribution maps and selected positions for different features, for the given scenario. The result of visual statistical learning-based familiarisation is a restructuring of the new page to the computed template design, consisting of positions for the matching features.

3.3.4 Principle IV: Visual Sampling Based on a Generative Cognitive Model

Here, a generative approach is used, where a model of visual search is used to generate gaze fixations as the simulated user is searching targets on a layout. The simulation integrates models of eye movements, visual short-term memory, and associative long-term memory, and proposes that visual search is the interaction of these three resources [69]. Given a user history with layouts, including locations of elements on the layouts, the model generates eye movement patterns as it simulates human-like visual search on the layouts. Using the model of eye movements [126], the simulation encodes elements on the layout until it has found the target. For each element, the encoding time is

$$T_e = K \cdot [-log(f)] \cdot e^{k \cdot \epsilon}, \tag{3.6}$$

where f is the frequency of the object, either supplied externally to the model or assumed to be uniform, ϵ is the distance of the target from current eye fixation, and K and k are scaling constants, adapted from the literature [126]. T_e increases exponentially as the target is further from the fixation, but the visual system may compensate this by initiating a fast eye movement to gaze closer to the target, with movement time of

$$T_s = t_{prep} + t_{exec} + D \cdot t_{sacc}, \tag{3.7}$$

where t_{prep}, t_{exec}, and t_{sacc} are constants from literature [126] and D is

the movement amplitude.

After the eye movement, the target needs to be encoded (3.6). However, if the encoding time is less than t_{prep}, then the target is encoded without the eyes moving. This creates an eye movement model where the eyes move only when the target is too far to encode from the current gaze point.

In addition to visual search, the model simulates learning of layouts by using an activation-based associative memory model [4], its location is stored in the memory storage. An activation strength of an association can be calculated based on the number of times the target has been found:

$$B_i = \ln(\sum_{j=1}^{n} t_j^{-d}), \tag{3.8}$$

where t_j is the time since the j:th time of finding the target i, and d is a decay parameter, set from literature [69]. As the model learns the layout, it can use the associative memory to recall the position of the target without having to visually search for it. The model is able to recall the position of the target, if $B_i > 0$, with noise added from normal distribution [69]. Recall time is:

$$T_i = Fe^{-fB_i}, \tag{3.9}$$

where F and f are scaling constants, set based on literature [69].

Longer history with a layout results in higher associative activations, and faster, more expert-like performance in finding targets. From the activations, the model can predict where the user will gaze,

given a layout and user history. Figure 3.4(c) illustrates activations, and predicted positions, for the given scenario. The result of this generative model-based familiarisation is a restructuring of the new page to the computed template design, consisting of positions for the matching features.

3.4 Familiariser: System Overview

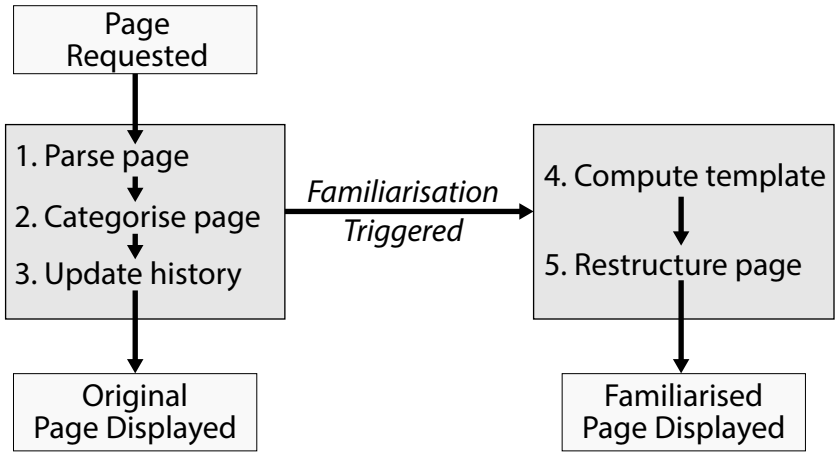


Figure 3.5: Familiariser Pipeline: When a page is requested, it is parsed and categorised, and the history is updated for visited pages. Once familiarisation is triggered, the template design is computed using an HVS principle, and restructuring is instantiated via layout optimisation prior to rendering on a web browser.

We implemented the concept of familiarisation in *Familiariser*, a browser-styled application that allows users to visit webpages and enables access to familiarised versions, adapted towards each user. As

illustrated in Figure 3.5, when a user visits a webpage, the following pipeline is executed:

1. **Parse page:** The page source (HTML) is parsed by the system. Key elements, representing high-level features, are detected and labelled.

2. **Categorise page:** The page is classified into a general website category (for example, shopping, banking, travel, etc.).

3. **Update history:** If the visited page does not exist in the history, it is added; if it was previously visited, the corresponding entry in the history is updated for the page. Additionally, familiarity scores, described in the previous section, are updated for every page in the user history.

   While familiarisation is disabled, the pipeline ends at this point, and the original page is displayed as is. Once familiarisation is triggered, the following steps (3 and 4) are executed to familiarise the page.

4. **Compute template design:** The history is filtered for pages belonging to the same category. Based on this, we compute the template design, to be used as the basis for restructuring the new page.

5. **Restructure page:** The new page is restructured using positional values of matching features from the template, and by repositioning the corresponding elements on the visited page. Overlaps may occur in the restructured page. Familiariser resolves

any such overlaps while attempting to best maintain relative alignments of elements on the page. This ensures a valid layout, without obscuring or omitting any contents. The familiarised page is finally displayed to the user.

Figure 3.6 and 3.7 illustrates two examples of results obtained by Familiariser, given a user history and original (unfamiliar) page, for each of the four presented principles.

In the remainder of this section, we describe the various components of the system in greater detail.

3.4.1 Page Parsing

A *feature* can be defined as a task-level element [151]. They are actionable elements, with a well-defined purpose, such as the logo, search bar, buttons or icons to login and access a user account, or the shopping cart on e-commerce sites, etc. Features can be composed of several low-level HTML DOM elements. For familiarisation, it is necessary to first detect these features on a given page. Prior works have explored element detection when underlying sources were not available. Prefab [34] presented a pixel-based approach to reverse-engineering the GUI. Sikuli [165] allowed users to take screenshots of widgets and elements, and use these for search and automation of visual interfaces. For webpages, however, the underlying source files are openly accessible. Previously, Webzeitgeist [81] used CSS selectors to detect common features. Familiariser parses the underlying DOM tree of a page, and analyses DOM tags, identifiers, and linked CSS classes, of elements to automatically detect features on the page. We use partial string match-

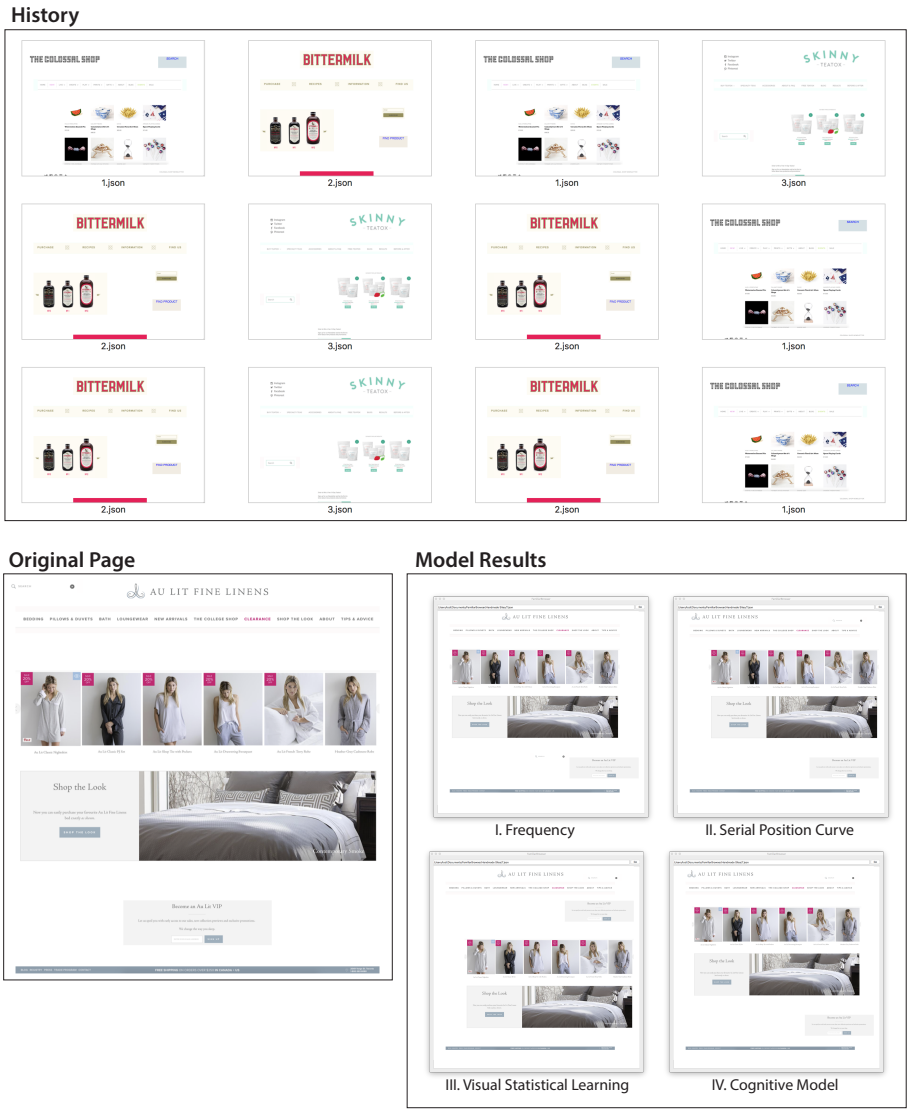


Figure 3.6: A visual comparison of the results after familiarising a page using the four presented principles. For a given user history, when a new page is visited, each of the models may produce varying results.

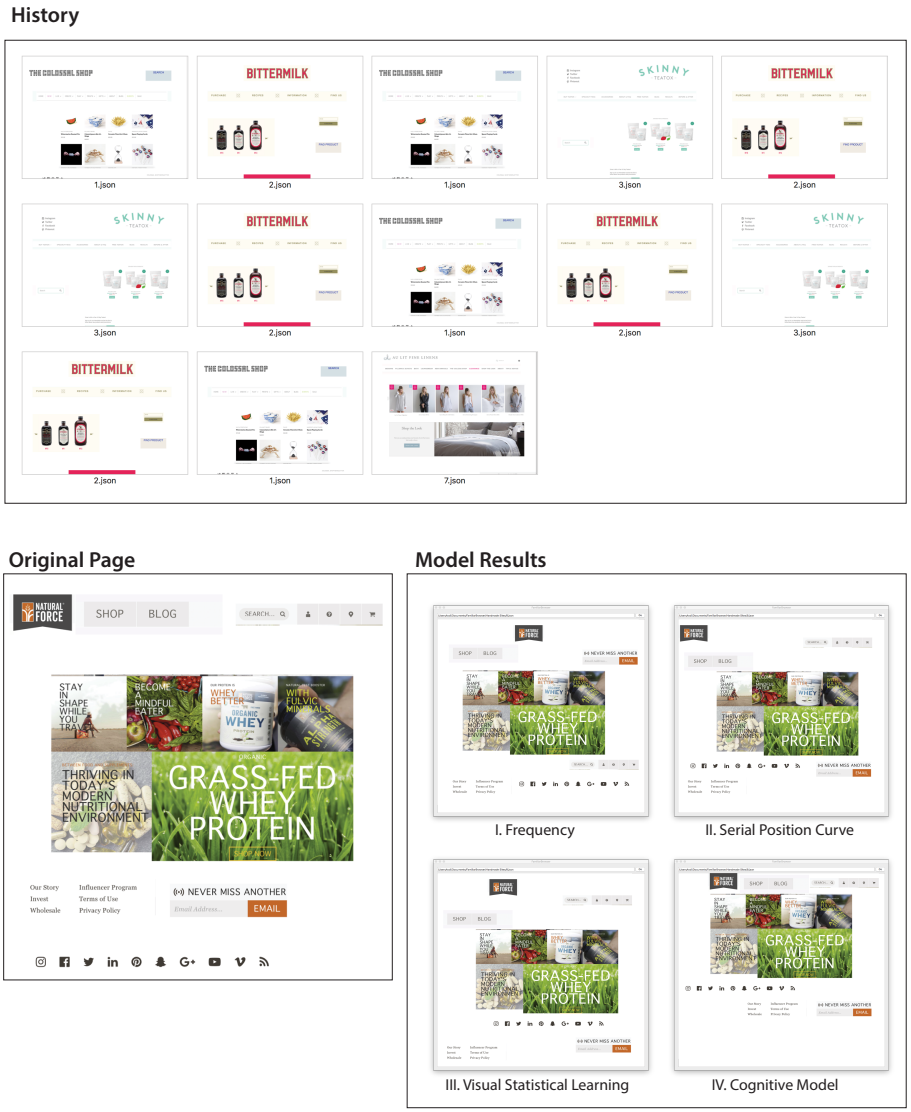


Figure 3.7: A second example of results from restructuring a new page. In this case, the user has visited additional pages, indicated in the updated history. The template is recomputed, and used to create familiarised results.

ing to detect commonly-used names, and map them to corresponding features. A feature can either be a leaf node in the DOM tree, or a compound element consisting of several smaller elements. The detection of features relies on appropriate naming and tagging of underlying HTML elements, and this can be a limiting factor in finding all matching features across different pages. Feature recognition could be improved by employing other computational methods, such as image recognition or machine learning, but this is out of the scope of this work.

3.4.2 Page Categorisation

A *category* defines a group of pages that semantically or functionally belong together, and are used in similar ways or for similar tasks. Since features on different categories of websites tend to have different properties or aesthetics, it is important to segregate them. Therefore, each page is assigned a textual category label once it has been parsed. Familiariser uses HTML tag annotations to categorise visited pages into general categories. Automatic categorisation can be improved by using other strategies such as topic modelling [17], or by matching features found across pages.

3.4.3 Usage History Updates

For each user, page visit history is recorded, and usage-based metrics are updated each time new pages are visited. For each page, a unique entry in the history is maintained, and includes information related to time spent on the page (duration), frequency of visits, recency, and

order of visits. When an existing page in history is re-visited, the frequency (f_{page}) is incremented, and the recency of that page is reset such that it is the "most recent" page (i.e. $r_{page} = 0$). For every other page in the history, the recency (r_{page}) is incremented, thus decreasing the recency score (S_r).

3.4.4 Template Computation

All pages in the history are considered to compute the familiar layout design. For web browsing specifically, different categories of websites have distinctly different features. To avoid mismatch in domain, website categorisation can be performed as an intermediate step, and only pages within the same category as the currently-visited page are considered. For example, if the user intends to visit a shopping website, only pages from the shopping domain are considered while computing the base layout design.

Principles I and II consider a single page as the template. Familiarity scores are calculated accordingly, and the highest-scoring page is selected. Matching features are extracted from the template, and their xy positions are used for the base layout design. Principles III and IV compute templates based on *all* pages in the user history. For principle III, probability distribution maps are created for features detected on all visited pages. Using these, the most likely position for each feature is estimated, resulting in a template where all feature are located appropriately. For principle IV, activation points for each feature are computed. The point with the highest activation, for a feature, is predicted to be the most familiar position, and hence used for the

template.

3.4.5 Target Page Restructuring

The template that is computed is used to restructure the newly-visited page. Features on the target page are matched to those on the template design, and repositioned accordingly. In [82], retargeting occasionally resulted in truncated or cropped content due to size mismatch. In our approach, while repositioning, dimensions of the original elements are maintained so as to avoid truncation of contents. This can however result in some overlapping or occlusion of elements.

Overlap Resolution

Overlaps are resolved by setting up a series of overlap-redressal rules. Examples of rules used in Familiariser are as follows:

1. Left-alignment or top-alignment of two (or more) elements should not be violated. If this is violated, a penalty is applied.

2. Movement of any element from its preferred (horizontal) location entails a penalty.

3. The vertical or horizontal sequence of any pair of elements should be honoured.

4. The canvas width should not be changed. The height may be changed if needed.

The rules are implemented using any standard integer-linear programming solver, resulting in a valid layout without overlaps.

3.4.6 Triggering Familiarisation

During the initial stages of browsing, the system updates its model, and displays the page in its original form. As users visit different pages, they gradually learn visual layouts of these, thus increasing chances of recall during future visits. Once a user is fluent with a small set of pages, familiarisation is triggered, and newly visited pages are adapted to match the computed template design. However, we need to determine the ideal moment to trigger familiarisation, to make it effective. There is a trade-off between familiarising too early and too late. If triggered too early, users may not have learnt visited pages sufficiently, thus future pages would not benefit from adaptation. On the other hand, if familiarisation is delayed, then users might have already been exposed to a large number of diverse designs, thus making recall harder. For the purpose of our study, we determined (by trial-and-error) that enabling familiarisation after 25 page visits was favourable, given that the number of unique pages was less than or equal to 5. For future systems, familiarity scores can be used to empirically determine when to trigger familiarisation, or this could be customisable per user.

With automatic (always-on) familiarisation, pages are automatically adapted when they are requested. Familiariser also supports manual (on-demand) familiarisation, where users can explicitly request for a familiarised version of page. Here, the original page is displayed by default, and a familiarised version is rendered only when

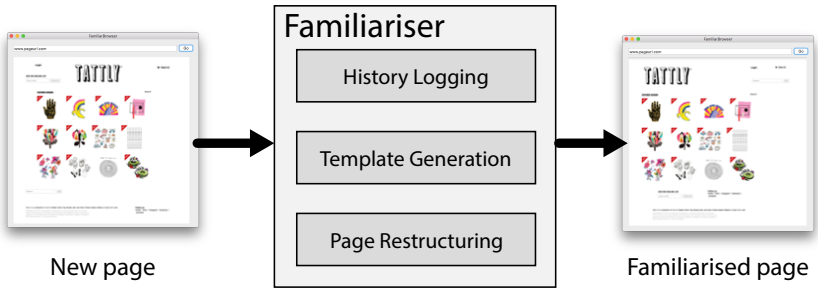


Figure 3.8: Main components of the Familiariser system. As users visit pages, their history is logged. This is used to model visual search and generate a template for the user. When a new interface is visited, it applies this template to restructure the page.

demanded. This is similar to how web services such as translation and reader-friendly modes enable users to access adapted versions of a page. Manual familiarisation obviates the need for a determining a fixed point at which adaptations are triggered.

3.5 Architecture and Implementation

The front-end of Familiariser is implemented in Swift, on MacOS 10.13, as a standalone browser-styled application. In the back-end, the system consists of 3 main components: logging user history, generating a template, and restructuring the page, illustrated in Figure 3.8. The details for each of these follows.

3.5.1 Logging User History

The system maintains a persistent history file for the user. Every time a new page is visited, custom JavaScript code parses the page source, and converts the page into a flattened JSON file. The JSON file is parsed to construct a webpage object with the following internal model:

1. *URL (String):* Unique address identifying the website.
2. *Category (String):* The general category of websites to which the page belongs, for example, shopping, banking, travel, social networks, etc.
3. *Page Elements (Array):* Content elements appearing on the webpage. Each element contains absolute (xy) positions as they appear on the browser, feature name, and other element-specific properties, such as tags and CSS styles, required to recreate the page.
4. *Primacy (Integer):* The visit order number indicating the primacy of the page.
5. *Frequency (Integer):* Number of times the page has been visited.
6. *Recency (Integer):* A number indicating how many other pages have been visited since the last visit.

When the user completes a page visit, either by navigating to a different page or by quitting the application, a new page record is added sequentially to the user history. This record contains an identifier for the webpage, a pointer to the internal webpage object, a timestamp indicating when the page was visited, and time duration of the visit.

This sequential history contains all information required to model a user's familiarity.

3.5.2 Generating a Template

When familiarisation is triggered, the system filters the history for all pages belonging to the same website category, and creates a familiar template unique to each category.
For *frequency-based familiarisation (I)*, the system finds the webpage with highest frequency in the user history, and selects it as the template.
For *serial-position curve (II)*, it calculates the familiarity scores for each webpage in the history, and selects the page with the highest score.
For *visual statistical learning (III)*, the system initialises an empty 2×2 array, with pixel dimensions of the largest webpage in the history. It iterates over all pages in the history, to create probability distribution maps by assigning appropriate pixels a weight, when a feature is found. Next, for each feature, it iterates over the respective map, and finds the pixel with the highest value. The final template consists of (x,y) points with highest probabilities for all found features.
The *cognitive learning model(IV)* is implemented using Common Lisp. Each time familiarisation is required, a CSV file containing relevant history information, including access timestamp, duration, and linked JSON file name, is generated. The CSV file and required JSON files for all webpages are passed as input to the learning model. As output, the model generates a new CSV file containing activation points (x,y) for all detected features, along with confidence values, and returns this to

Familiariser. Familiariser uses this to select the desirable positions for features, to generate the template.

3.5.3 Restructuring the Page

Given the template, Familiariser automatically restructures a newly-visited page by repositioning matching features at locations founds on the template design. Any unmatched features are then positioned at their original location. Next, overlaps in layout elements are resolved by passing a file containing element positions to a Java application. The application applies overlap-redressal rules to each element. We use an IBM CPLEX linear programming solver to optimally resolve any detected overlaps. The solver returns a corresponding file with resultant element positions. This is used by Familiariser to generate the final valid layout, which is then displayed to the user in the browser-based application.

3.6 Evaluation

The goal of our evaluation is to verify the concept of familiarisation, and the presented system. We seek to answer the question: *Does familiarisation improve performance in web browsing for end-users?* To this end, we conducted a comparative user study, and report on quantitative results. We compared original (unmodified) designs against familiarised designs in a study where users were asked to point-and-click on different features on the displayed page. We used the visual statistical learning principle (principle III) for familiarisation as these selection

tasks were quite brief in duration. As dependent variables, we analysed visual search time, approximated by pointing time, and number of eye-gaze fixations per target feature. Comparing these two cases enables us to evaluate the potential effects of familiarisation during real-world usage. For the test case, we chose the domain of shopping websites, a frequently used category of webpages that are also plenty in number. These websites typically contain similar features yet vary vastly in their layouts and presentation of these features. This makes them a good test candidate for Familiariser, and provides a realistic use scenario for the study.

3.6.1 Study Tasks

For the study, participants were given the task of selecting (clicking on) a feature element on the displayed webpage, in a browser-based application. Webpages were selected randomly, from a dataset of 30 shopping sites. The target feature was also selected randomly from the page, and included commonly occurring elements, such as logos, navigation menus, search boxes, among others. Participants were requested to perform tasks quickly yet accurately as possible, and avoid unnecessary pauses. As tasks were performed, the software logged mouse movements, click events, and eye gaze information, including eye position (averaged) and fixations.

3.6.2 Apparatus

A 13" MacBook Pro with Retina Display (2560-by-1600 pixels), running MacOS 10.13, was used for the study. The browser-based Famil-

iariser software was displayed as a full-screen application. We selected 30 shopping websites to create the dataset, excluding commonly-used shopping pages to avoid bias. Webpages used in the dataset were rendered offline to avoid delays. Tasks were completed using the in-built trackpad. All cursor motions and events were logged by the study software. For eye-tracking, an EyeTribe Tracker[1] was used, along with a custom Python program to log gaze positions and fixations.

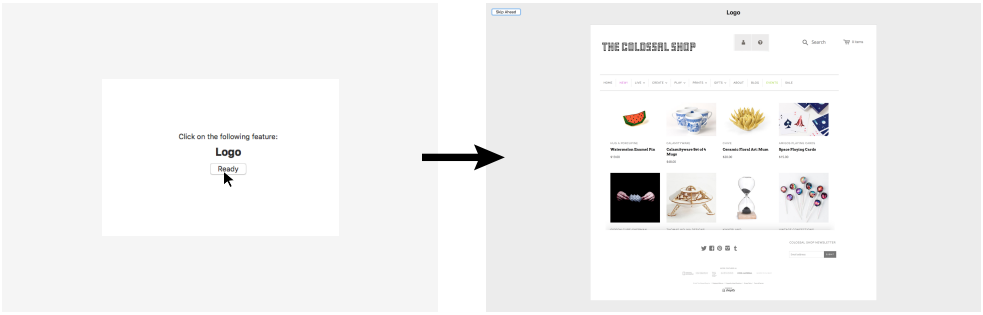
3.6.3 Participants

We recruited 16 participants, aged 21 to 36 (mean 29), for the user study. Participants reported to have no visual impairments, or corrective glasses. All participants reported frequent web usage (daily), and also reported recent exposure to shopping websites.

3.6.4 Method

During the experiment, participants were exposed to a set of different websites. Before a webpage was displayed, they were presented with the task of selecting (clicking on) a particular feature element (e.g. logo, main menu, search box) on the page. The target feature element was selected at random from the page to be displayed. The timed trial started once the participant confirmed they were ready, by clicking on a confirmation button. This button was consistently placed at the centre of the screen, ensuring a constant starting point for the cursor during all trials. Figure 3.9 shows screenshots of this setup.

[1] www.theeyetribe.com



Task Stimulus Page Displayed

Figure 3.9: Screenshots from one trial of the study. The user is first shown a stimulus, with the task description. The task is started when the user clicks on the "Ready" button, thus positioning the cursor on the screen centre. A page is then displayed; the task is completed when the user successfully clicks on the target.

The experiment was divided into two phases: *Learning Phase* and *Test Phase*, as described below.

1. Learning Phase

For each participant, the system selected 5 webpages, at random, from the dataset of 30 pages, for the learning phase. The learning dataset thus varied for each participant. During each trial, a page from this subset was selected at random, and the participant was asked to point and click at a randomly chosen feature on the page. A total of 25 trials were performed during this phase, and this was used to construct the 'usage history' for the participant.

2. Test Phase

Once the participant had experienced this subset of pages in the training phase, the experiment moved on to the test phase. The 5 pages used in the learning phase were stored in the user history, and excluded from the test dataset. Similar to the learning phase, participants again browsed to randomly selected pages, and were asked to select a target feature on the page. During the test trials, either the original (unmodified) page or a familiarised version was presented to the user, chosen at random. All participants were exposed to both conditions, original and familiarised, resulting in a within-subject study design. Participants performed a total of 100 timed trials during the test phase.

The average duration of the study was approximately 30 minutes for each participant.

3.6.5 Results

We created a linear mixed model with search time as dependent variable, page type (Original vs. Familiarised) as fixed independent variable, and participant id and page URL as random variables. Grand mean search times were Original = 2.8 seconds and Familiarised = 2.5 seconds. The difference was statistically significant, $t(663) = 5.3, p < .001$, with model $F(1, 663) = 28.5, p < .001$. In standardised units, the benefit of familiarised layout was $\beta = 0.35$.

We compared similarly the number of fixations per target. Grand mean fixation counts were Original = 3.4 and Familiarised = 2.6. The difference was statistically significant, $t(596) = 4.7, p < .001$, with

	Visual Search Time	Fixation Count
Original	2.8 seconds	3.4
Familiarised	2.5 seconds	2.6

Table 3.1: Summary of results for average visual search time and fixation count per target feature.

model $F(1, 596) = 22.3, p < .001$. In standardised units, the benefit of familiarised layout was $\beta = 0.35$.

The above results indicate that familiarisation improved user performance by reducing both visual search time and number of gaze fixations. Our study evaluates familiarisation using the visual statistical learning principle. A comparison of all four principles requires a more extensive study, and is left as subject of future work.

3.7 Discussion

3.7.1 Summary

Our work on familiarisation of visual designs situates itself in the field of automatically generated user interfaces that adapt to individual users. The cost of adaptation is often a concern for adaptive user interfaces. Our work circumvented this as familiarisation restructures only new and unfamiliar designs, at use-time, instead of continuously adapting or modifying frequently-used layouts in real-time. One could criticise our approach for compromising brand identity, or undermining the designer, by modifying designs. However, we argue that usability of an interface supersedes these aspects for the end-user.

Additionally, Familiariser addresses this by allowing users to optionally view either original or familiarised versions of designs. Commercial browsers have also used such techniques to improve usability of webpages. For instance, 'reader-friendly mode' on browsers allows users to switch between the original page and a version enhanced for reading.

Familiarisation deals with concepts of recall and visual learning to make interfaces appear closer to each user's expectations. We explore four principles of familiarity, inspired by the human visual system, and grounded in literature. Familiariser implements these in an end-user system that captures users' history, and restructures newly visited pages based on automatically generated familiar layout designs. Results from the empirical study provide evidence for our approach. Familiarisation significantly improves visual search time by over 10%, and reduces the number of fixations by over 23%, while searching for features on a given design.

3.7.2 Revisiting the Research Question

To improve placement of elements on GUIs, this chapter aimed to investigate user-sided techniques. The research question, as stated in the introduction, is:

*How can we **adapt** graphical interfaces for individual users' by automatically placing elements at familiar locations, at use-time, such that they are consistent with a user's mental model and enable faster visual recall?*

To this end, I presented familiarisation as an approach to automatically adapting user interfaces at use-time. By logging user history, and

modelling familiarity based on visual search, we could automatically adapt and improve placements on graphical layouts, in a principled manner, for each user.

The main technical contributions of this chapter towards realising the bigger goal of improving GUI layouts are:

1. *Familiarisation principles* to compute a familiar template based on a user's visual history.
2. *The familiarisation pipeline* to log user history, model familiarity, and restructure unvisited layouts.
3. *Familiariser*—a browser-based implementation that records user history, applies the principles to generate a template, and automatically restructures webpages to generate a valid familiarised layout.

Our evaluation with users illustrates the benefits of familiarisation on user performance during a typical web browsing task.

3.7.3 Principles for Use-Time Placement

The key design principles derived from this work, towards the goal of improving placement on GUIs at design-time are:

1. **Capture detailed usage history** for each user by logging interactions.

2. **Model visual familiarity** by taking into account different statistical and perceptual aspects of usage.

3. **Avoid cost of adaptation** by only modifying new, unvisited layouts at use-time.

4. **Ensure completeness and legibility** of modified UI layouts by resolving any structural inconsistencies before rendering.

3.7.4 Limitations and Next Steps

There are certain limitations for applying familiarisation universally, as it requires (1) logging of a user's history of seen layouts, (2) detailed information and representation of layouts, (3) just-in-time computations of templates, and (4) instantiation of page restructuring in runtime, prior to rendering it on a display. By exploring a range of principles for familiarisation, we provide alternatives that enable systems to circumvent some of these limitations. The basic frequency-based approach (principle I) is straightforward to implement, and requires minimal user history information. Serial-position curve (II) requires some additional usage information, and requires calculations of various scores to select a page as the template. The feature-based principles (III and IV) require detailed information about the interface layout, and are computationally more expensive, but offer more accurate representations for a user.

While this chapter, and our implementation, focuses on repositioning of features, future familiarisation efforts can apply our work to address other interface aspects, such as colours, fonts, and other visuoperceptual properties. Such properties are often position-agnostic, and thus for these page-wise familiarisation (I or II) is preferred. Apart from addressing additional interface properties, usability and expe-

rience with user interfaces can be further improved. Future systems can explore a dual-optimisation strategy, where the familiarity model can be combined with other predictive models of human perception. Additionally, more interfaces can be covered by applying the concept of familiarity to a variety of mediums (digital, physical, hybrid). For instance, it can be possible to familiarise physical interfaces, adapting them to resemble previously encountered digital or physical interfaces. Finally, by providing a formal model of familiarity, we can implicitly gain an understanding of "unfamiliarity". This opens up possibilities of other applications such as 'unfamiliarisation' or diversification of interfaces. This could encourage alternative design goals such as exploration, or be used to draw users' attention to certain elements.

3.8 Acknowledgements

A part of this research was conducted during my second internship at the User Interfaces Group, at Aalto University. The project was partially funded by the Academy of Finland project COMPUTED and the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement 637991).

Apart from my co-authors, Jussi Jokinen, Kris Luyten and Antti Oulasvirta, I would also like to thank thank Niraj Damaya for code related to overlap resolution, Markku Laine for his useful comments, and study participants for their time and involvement.

PART II

Facilitating Placement on Post-WIMP User Interfaces

Post-WIMP user interfaces [149] go beyond the typical graphical interfaces by integrating richer interactive capabilities into the interface. Typical examples of such interfaces include gestural interfaces, tangible UIs, multimodal interfaces, and ubiquitous computing. In the second part of this dissertation, I discuss some of the challenges for post-WIMP UIs, and investigate how we can facilitate, or enable, the placement of interactive elements on such interfaces.

Chapter 4

Facilitating Placement of Interactive Electronics on Post-WIMP Interfaces

In the first part of this dissertation, I discussed strategies for placing interactive elements on graphical interfaces. The focus was on improving GUI interface layouts, to improve their aesthetics and performance. In the next part, I address placement issues related to post-WIMP interfaces. Such interfaces often add a physical component by, for example, applying novel interaction techniques, or augmenting interfaces with interactive electronic elements such as sensors and actuators. As a result, these interfaces are also technically more complex than GUIs. The focus of this part, therefore, is on *facilitating* the placement of such interactive elements on post-WIMP user interfaces.

First, I investigate the placement of electronic components, such as sensors and actuators, onto physical post-WIMP interfaces such as

interactive paper. Paper is a widely-used physical medium, but has traditionally been largely static in nature. Recent technological innovations such as printed circuits have enabled the integration of electronics into paper substrates, opening up the possibilities of making it a dynamic medium. However, placement of interactive elements on paper is a challenge. It typically requires knowledge of electronics as well as programming, making it inaccessible to *non-experts*. In this chapter, I address challenges to placing interactive electronic elements on paper, and present *PaperPulse* as a viable approach, enabling non-experts to create such post-WIMP interfaces. PaperPulse streamlines the process of creating paper interfaces, providing users with an integrated workflow to place interactive components onto paper substrates, and specify the desired interactivity.

This chapter is based on the following publications:

1. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2457–2466. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702487
2. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach to fabricating interactive paper. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 267–270. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3146-3. doi: 10.1145/2702613.2725430
3. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paper-

pulse: An integrated approach for embedding electronics in paper designs. In *SIGGRAPH 2015: Studio*, SIGGRAPH '15, pages 3:1–3:1. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3637-6. doi: 10.1145/2785585.2792694

4. RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *ACM SIGGRAPH 2015 Posters*, SIGGRAPH '15, pages 9:1–9:1. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3632-1. doi: 10.1145/2787626.2792650

4.1 Introduction

Post-WIMP user interfaces go beyond traditional GUI by integrating new forms of interactivity and interactive elements, such as novel sensing techniques, electronic components, interactive tangibles, among others. Interactive paper is one such category of post-WIMP interfaces, which can be realised by placing electronic components onto the physical mediums.

4.1.1 Placing Electronics on Post-WIMP Interfaces

To make electronics available for non-experts, construction kits targeting programmers, such as .net gadgeteer [60] or Phidgets [48], or non-programmers, such as littleBits [13] provide modules to rapidly build hardware prototypes. However, these kits are often bulky and expensive. Thus, for instance, it is not feasible to create interactive greeting cards that can be handed out, or games that are seamlessly integrated

into paper, like the one illustrated in Figure 4.1e. In a similar vein, design tools, such as Midas [129], d.tools [57], Examplar [56] or Boxes [64] make it easier for users to link sensor data to application logic through programming by demonstration. However, these tools require users to have some exposure to programming languages. Additionally, they do not allow for standalone systems since they assume hardware sensors to be connected to a desktop computer at all times.

4.1.2 Interactive Paper

Recently, there has been a growing interest in different fields and communities (e.g. research, maker movement, engineering and marketing) in making paper interactive by augmenting it with electronics. Interactive input and output components such as buttons, sliders, LEDs, and buzzers, among several others, can now be placed on flat paper substrates, and connected using printed circuits. This makes it possible to produce low-cost paper versions of PCBs in lab environments [71] and bring liveness to paper artefacts such as books [117, 116] and posters [135]. Although advancements in fabrication tools for electronic circuits, such as conductive pens, threads, inkjet printers [71] and vinyl cutters [129] make it accessible for many people to build these paper circuits, a vast majority lacks expertise in electronics and programming to make paper interactive using electronic circuits.

4.1.3 Research Question

While circuit building and programming skills can be acquired by non-experts through workshops and tutorials [117, 98], adding electronic

circuits onto post-WIMP interfaces, such as paper or wearables, is not yet as convenient as adding visual designs on paper with common graphical software tools, such as Illustrator or InDesign.

To facilitate the placement of interactive electronic elements, such as sensors and actuators, on physical post-WIMP interfaces, this chapter discusses the research question:

How can we facilitate non-experts in placing interactive electronic elements on commonly used physical mediums by obviating the needs for programming and electronics skills?

To this end, this chapter investigates an end-to-end workflow (illustrated in Figure 4.1, and presents *PaperPulse* as a design tool that enables users to place interactive electronic elements onto paper interfaces.

4.1.4 PaperPulse: Placing Electronic Elements onto Paper Interfaces

PaperPulse is a design tool that assists in, and automates, parts of the design, programming, and fabrication of electronic paper circuits. With PaperPulse, non-expert users make standalone interactive paper artefacts in which electronic components are seamlessly integrated in visual designs. An end-to-end workflow enables users to enrich visual designs with interactive elements without the need for programming the application logic, or constructing electronic circuits manually. The remainder of this chapter describes the PaperPulse workflow and design tool in detail, and illustrates some sample paper interfaces that can be realised using our tool.

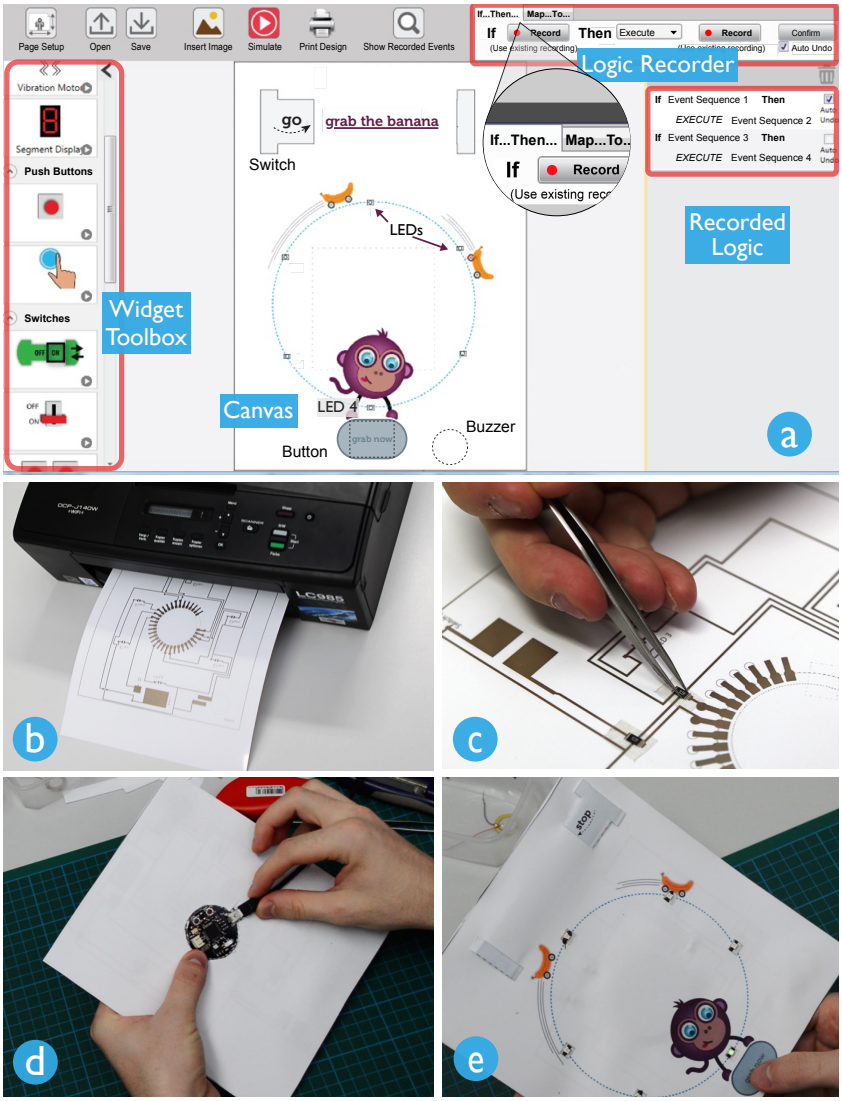


Figure 4.1: The *PaperPulse* workflow streamlines the entire process of creating interactive paper artefacts. (a) Design and specify logic; (b) print sheets; (c) assemble; (d) upload generated program to microcontroller; (e) final paper artefact.

4.2 Background

The work presented in this chapter builds on fabrication techniques for designing electronic circuits and design tools for sensor-based interactions.

4.2.1 Fabricating Electronic Circuits

Modular electronic construction kits, such as Little Bits [13], .NET Gadgeteer [60], Phidgets [48], Calder toolkit [85] made it easier and thus more accessible for non-experts to build electronic circuits. To preserve the aesthetic and expressive qualities that traditional crafting materials provide [98, 113], researchers have investigated different techniques to integrate flexible circuits directly into substrates using copper tape [117], conductive ink [98], threads [113] or fabrics [116]. These techniques have been used for different purposes, for example, to electronically augment pop-up books [116, 117], design interactive invitation cards, posters and paper headphones [135], and enrich origami and paper sculptures [128]. To ease and speed up the process of fabricating electronic circuits, researchers explored various techniques, such as chemical sintering with off-the-shelf inkjet printers [71], cutting copper foil with a vinyl cutter [129], drawing conductive traces with a plotter [37], integrating circuits directly in the paper making process [30], and by making adhesive [61] stickers with integrated PCB's available.

Although these efforts make it easier to fabricate electronic circuits on materials such as paper, it still requires users to have basic knowledge of electronics, something the test subjects in some of the previ-

ously discussed platforms acquired through workshops [98, 117] and online tutorials [113]. Similar to Midas [129], PaperPulse automatically generates electronic circuits with step-by-step instructions to assist the assembly process.

PaperPulse thus shares inspiration with Midas, but it offers important contributions beyond this work. First, artefacts designed with Midas are not standalone systems and need to be connected to a desktop computer at all times. Secondly, Midas only supports capacitive sensor pads. Also, the extensive logic support in PaperPulse is not offered by Midas. Unlike Midas, PaperPulse is not limited to planar circuits and produces much smaller and less fragile circuits.

4.2.2 Design Tools for Sensors-Based Interactions

To make it convenient for programmers to work with electronic components, researchers developed well-defined programming interfaces for micro controllers [60, 98] as well as for specific electronic I/O components [12, 48, 85]. Researchers have also developed various visual programming approaches to empower non-programmers make sensor-based interfaces. Some of these approaches are analogous to building blocks, such as ScratchForArduino[1] and eBlocks [92]; other systems support basic functionalities by analysing handwritten keywords [18].

Instead of specifying logic visually or textually, programming by demonstration generates program logic under the hood by observing examples. This approach has been used to record simple keystrokes

[1] S4A: Scratch For Arduino. http://s4a.categories

and mouse clicks and replay them when an input event is recognized [64, 129]. Other systems record higher dimensional signals, such as sensor data from accelerometers [56] or cameras and microphones [33] and generalize rules using machine learning techniques. To support more complex sets of rules, demonstration techniques are also used to define transitions in statecharts [57].

The logic supported by our programming by demonstration approach, Pulsation, is closest in spirit to PICL [41]. Both Pulsation and PICL support discrete as well as continuous events. In contrast, Examplar [56] and d.tools [57] focus solely on extracting discrete events from continuous input streams. As such, there is no direct support for mapping a continuous input signals (e.g. a potentiometer) to a continuous output signal (e.g. an LED). Although Pulsation is a software platform and not a hardware platform as PICL, there are also important differences in logic: PICL only supports a single input and output signal whereas Pulsation has extensive support for defining time-related relations between multiple input or output signals. This makes it possible to specify that multiple actions need to happen simultaneously, sequentially or after a certain amount of time. Furthermore, Pulsation also allows to map derived signals as explained in section *Map-To Rules*.

4.3 PaperPulse: An Overview

PaperPulse enables users without a technical background to make traditional designs on paper interactive by seamlessly placing I/O components and microcontrollers. We believe that these components will

soon become cheap enough to enrich every paper design, from books, posters, and business cards to ephemeral packaging material and flyers.

Figure 4.1 shows how PaperPulse streamlines the design and fabrication process of interactive paper artefacts. (a) The user places interactive elements (e.g. push buttons, sliders, LEDs, microphones) on the visual design, and specifies the logic between components by demonstration. (b) PaperPulse generates different layers, consisting of visual elements and electronic circuits printed using an inkjet printer filled with conductive ink [71]. (c) By following step-by-step instructions, and placing electronic components on the printed circuits, the user assembles the different parts. (d) Next, PaperPulse generates code that can be directly uploaded to the microcontroller attached to the paper. (e) The design can now be used as a standalone interactive artefact.

4.3.1 PaperPulse Essentials

Although electronic circuits generated with PaperPulse can be fabricated using various techniques (e.g. a conductive pen, vinyl cutter), the circuits are optimized for printing on resin coated paper using a conductive inkjet printer [71]. To finalize the printed circuit, electronic components, such as resistors, buttons, switches, and LEDs, are attached using ECATT-tape[2] or conductive paint. PaperPulse supports both Netduino[3] and Threadneedle[4] microcontrollers. Pins on the Netduino connect to paper circuits using bulldog clips. In contrast,

[2]Electrically Conductive Adhesive Transfer Tape
[3]www.netduino.com
[4]modlab.co.uk

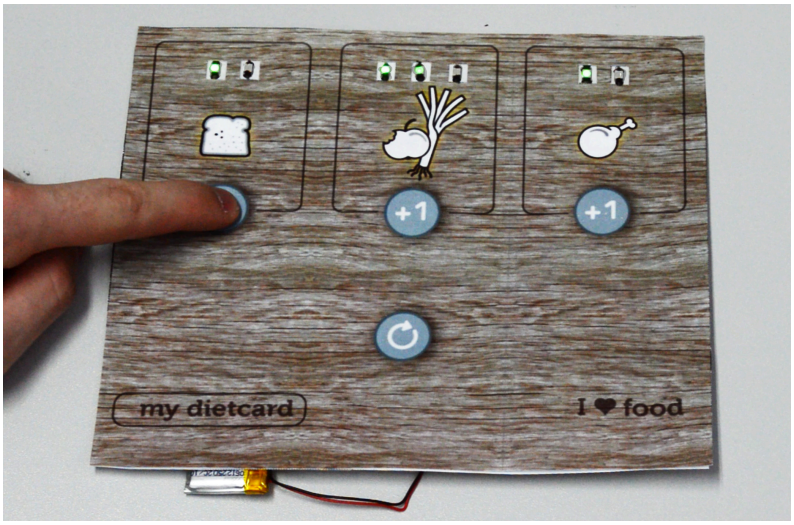


Figure 4.2: An interactive diet card to keep track of the user's daily food consumption. Touch buttons are used to turn on LEDs, which indicate the currently consumed amount. A 'reset' button clears the current state, allowing the card to be reused.

Threadneedle exposes flat connection pins that seamlessly connect to the circuit printed on paper (Figure 4.1d).

4.3.2 Walkthrough: A Diet Tracking Card

The following walkthrough illustrates the process of placing electronic elements, and constructing an interactive diet tracking card with PaperPulse (Figure 4.2). The card consists of three food categories, with LEDs to indicate daily consumption. A user can track their consumption by pressing on three touch buttons, one for each category. A reset button allows them to clear the current progress, and to reuse the card.

Step 1: Designing the Interactive Paper Layout

The user starts by specifying the dimensions of the paper design. PaperPulse then allows to import pre-designed visual elements (i.e. images) and to arrange them onto the canvas (Figure 4.1a). Next, the user places interactive components (seven LEDs and four push buttons), available in the widget toolbox (Figure 4.1a), onto the design. To give users a better idea of the look and feel of different components, tooltips with video previews [52] are available in the widget toolbox.

Step 2: Defining and Verifying Logic Iteratively

The user begins by recording the logic for connecting the 3 buttons to the corresponding LEDs:

(a) The user starts a new input recording in the *map*-part of the logic recorder.
(b) On the canvas, she demonstrates one of the buttons being momentarily pushed.
(c) She connects this to an output recording in the *to*-part of the logic recorder.
(d) She sequentially selects each of the relevant LEDs, and demonstrates their brightness changing to 100%.
(e) In the logic recorder, she specifies that the *repetitions* of button press are mapped to the progress of the LEDs.
(f) Similarly, the user repeats the steps to specify the logic for all 3 buttons and respective LEDs.

To verify the recorded rule, the user starts the simulator to interact with the widgets and observes the corresponding output. By observing fulfilled conditions and executed actions in the *Debug View*, the

user can identify possible mistakes in the recorded rules.

When the 'reset' button is pressed and released, all LEDs are turned off, and the diet card is reset to the starting state. Next, the user records this logic:

(a) The user starts a new input recording in the *if*-part of the logic recorder.
(b) She records the reset button press and release action.
(c) She then records a new output recording to reset the progress of all 3 buttons.

Step 3: Printing and Assembly

Once the design is complete, the user specifies the position of the microcontroller, and verifies that electronic connection pins for the widgets do not overlap. She adjusts widgets (e.g. position, size or orientation) if necessary.

The printing process starts by generating: (1) An electronic circuit that connects widgets to pins on the microcontroller while limiting the number of intersecting circuit traces; (2) PDF files consisting of the electronic circuits, widget-specific assembly lines (e.g. cut lines, fold lines), and visual elements; (3) Microcontroller code; (4) A customized tutorial to guide the user through the printing, deployment, and assembly.

Following the automatically generated tutorial, the user is instructed to print the generated PDF files (Figure 4.3) on three sheets of paper, using a conductive inkjet and a colour printer, as required. She then uses ECATT tape to attach bridges (zero-ohm resistors) at intersecting traces that could not be resolved by the auto-routing algorithm. The remainder of the tutorial provides instructions to cut,

fold and glue layers of paper, attach electronic components, such as LEDs, resistors, and attach the microcontroller and upload the generated code.

As shown in Figure 4.2, the resulting end-product can now be used as a standalone paper interface by connecting a battery.

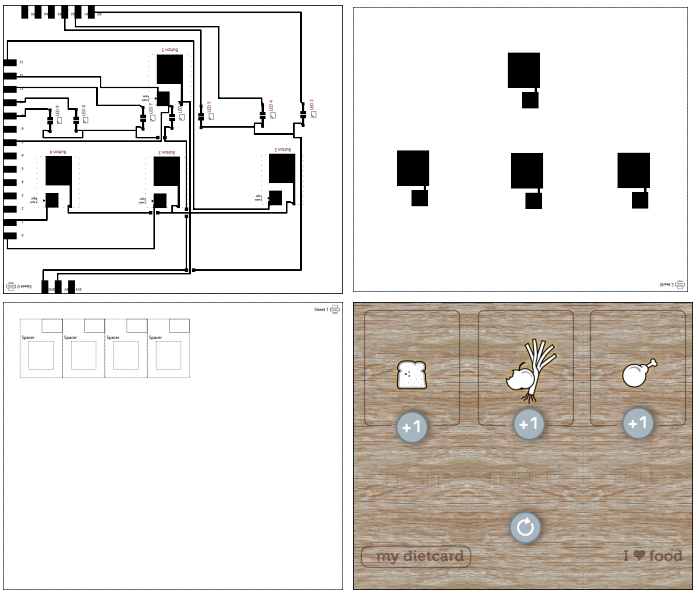


Figure 4.3: Generated sheets with circuit designs, graphics, and widget-specific traces. The user follows generated instructions to print and assemble the paper interface.

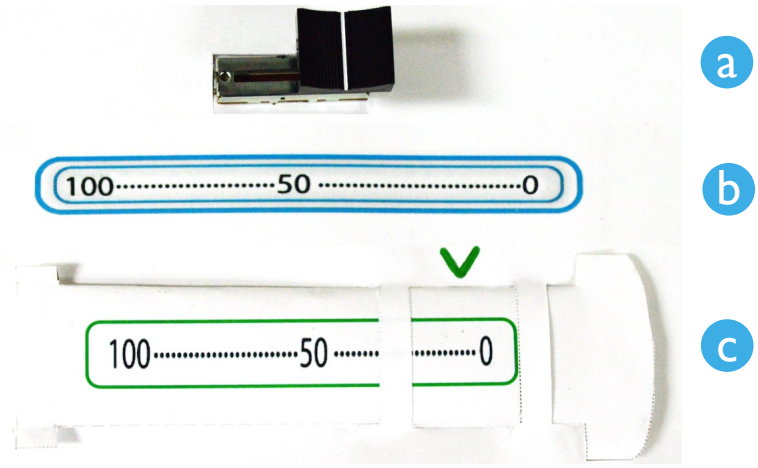


Figure 4.4: The three families of PaperPulse widgets: (a) Off-the-shelf slider; (b) Paper-membrane slider; (c) Pull-chain slider.

4.4 PaperPulse Widgets: Interactive Electronic Elements

To support a diverse range of interactive elements, suitable for paper designs, we present three families of standard widgets to realise basic controls such as push buttons, switches, sliders, and radio buttons. Each family is unique in its own way, and provides some strengths to distinguish itself from the others. Figure 4.4 illustrates how each approach realizes a linear slider.

4.4.1 Design Challenges

Our three families of standard widgets draw inspiration from the work by Qi and Buechley [116, 117] and Kickables [130]. However, design-

ing reusable widgets that can be printed turned out to be non-trivial: How can we ensure the continuity of the brittle circuit traces over folding structures? How can moving parts be powered? How can the firmness be increased and widgets made durable?

The three widget families consist of a different number of layers. To allow widgets of all three families to co-exist in a single design, we devised a uniform layering approach: a base layer, a widget-specific layers (where needed), and a top layer. This layering approach is also vital for the seamless integration of electronics and visual elements, since all conductive traces are concealed. Every widget design ensures that all conductive lines are traced back to the base layer, which is connected to the microcontroller.

4.4.2 Off-the-Shelf Widgets

PaperPulse currently supports eight off-the-shelf input sensors and four output components (Figure 4.5). Some components expose flat connection pins on the bottom (SMDs[5]) and therefore are attached directly to paper using ECATT-tape. Components having very small connection pads or regular connection pins (through-hole components) are first attached to a custom-built flexible PCB substrate that exposes large connection pads to the paper circuit, similar to Circuit stickers [61]. Alternatively, through-hole components can be extended with crimp terminals.

Although off-the-shelf widgets require only little manual assembly, they have a fixed design and often protrude from the surface. When

[5]Surface Mounted Devices

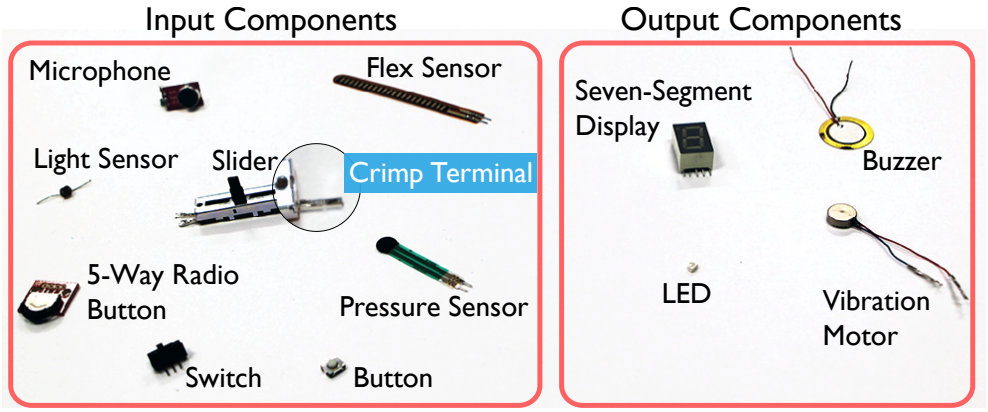


Figure 4.5: The off-the-shelf widgets currently supported by PaperPulse.

augmenting paper designs with electronics, it is often desirable to resize components and integrate them seamlessly with visual elements on paper. This is accomplished with *paper-membrane* and *pull-chain* widgets.

4.4.3 Paper-Membrane Widgets

Figure 4.6 shows two paper-membrane widgets. The main design rationale behind paper-membrane widgets is to create an electronic circuit between the base layer and back of the top layer and separate them with thin air gap using a paper frame (widget-specific layer) that serves as a spacer (Figure 4.6a). Pressing on the top layer connects it to the bottom, closing the circuit and thus realizing a push button. The top layer is powered from the base layer by connecting regions *Z1* and *Z2* using ECATT-tape.

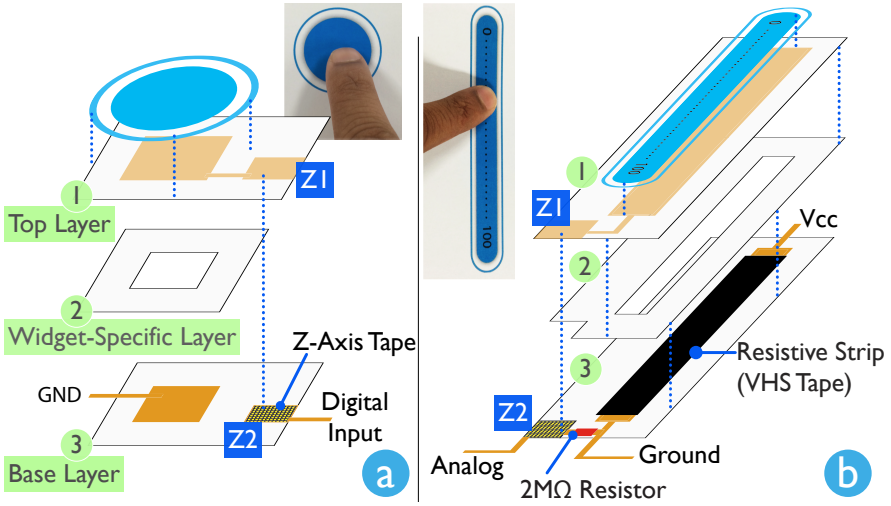


Figure 4.6: Design of paper-membrane widgets: (a) push button (b) slider.

Figure 4.6b shows the design of a paper-membrane slider in which the principle of a variable voltage divider is applied to measure the position where the top (wiper) and base layer make contact. To increase sensor resolution, the resistive strip should have a large resistance range. Although resistive strips can be printed (by reducing the opacity, and hence quantity of conductive ink) or drawn using graphite [62], we noticed that due to wear-and-tear the resistance of these strips often changes at frequently touched spots. For paper-membrane sliders, we therefore use resistive 8 mm VHS tape[6] as sensor strip, resulting in a more durable paper-membrane slider.

Paper-membrane widgets support radio buttons and switches by incorporating multiple paper-membrane push buttons in a single wid-

[6]Several other kinds of tapes could also exhibit linear resistance.

get with a shared software state. In contrast to off-the-shelf widgets, paper-membrane widgets are customizable. On the other hand, they do not offer tangibility. This is the essence of pull-chain widgets.

4.4.4 Pull-Chain Widgets

Pull-chain widgets draw inspiration from planar paper pop-up mechanisms [26]. Similar to off-the-shelf widgets, pull-chain widgets provide tangibility but at the same time do not protrude from the surface. Since they are designed entirely out of paper, pull-chain widgets are customizable and blend seamlessly into paper designs.

Although pull-strip mechanisms are traditionally used as sliding mechanisms [116], we see them as omnivalent pulling mechanisms in the same way as old-fashioned pull chains were used to control electrical appliances, such as light bulbs and fans. Figure 4.7 shows a pull-chain switch, slider, radio button and push button (using a crossing interaction technique [6]).

The mechanisms used for pull-chain widgets are optimized for tracking with electronic circuits printed on paper. These conductive traces are often brittle and cannot span across folded structures. As shown in Figure 4.8, the mechanical design of every pull-chain widget consist of (a) a folded tube structure with a hollow centre to ensure strength and rigidity during pulling and pushing motions, (b) slots to guide the pull-strip, (c) a wing tab to lock the pull-strip in place and (d) a pull-tab that functions as handle. The pull-strip itself is interwoven in the top layer. In combination with the tube structure, this provides sufficient pressure between the pull-strip and the base layer to ensure

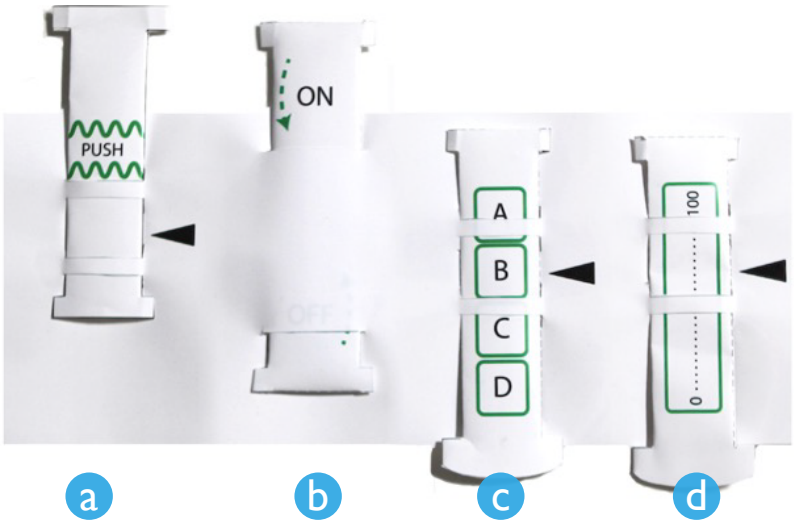


Figure 4.7: Pull-chain widgets supported by PaperPulse: (a) Push-button, (b) Switch, (c) Radio button, (d) Slider.

electrical connectivity, and at the same time provides an acceptable amount of friction to manipulate pull-chain widgets comfortably.

Figure 4.8 also shows the electrical circuit design specifically for pull-chain sliders. This consists of an analog sensor strip (8 mm VHS resistive tape) and winded circuit traces on the back of the pull-strip. Pull-chain radio buttons use the same approach but software thresholds are used to realize discrete states. In contrast, pull-chain push buttons and switches consist of conductive patches at specific spots that make an electronic connection when the strips are pushed or pulled. Push buttons, switches and radio-buttons usually employ mechanical detent mechanisms. These techniques however do not transfer to paper since paper is too fragile. To avoid undesired oscillations when

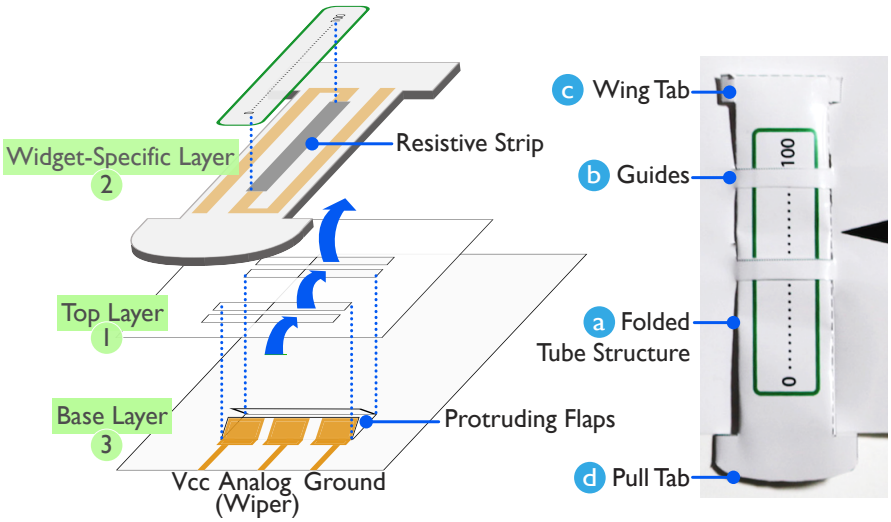


Figure 4.8: Design of pull-chain widgets: The widget-specific layer is interwoven into the top layer by passing it through four slots. Protruding flaps on the base layer also pass through these slots to ensure constant contact between the winding circuit traces on the pull-chain and the three pin connections on the base layer.

widgets are in between states, hysteresis and timeouts are used in software.

4.4.5 Summary of PaperPulse Widgets

In order to provide users a wide variety of widgets in PaperPulse, we presented three families of standard widgets. As shown in Table 4.1 each design offers its own strengths and limitations.

We distilled the paper-membrane and pull-chain widget designs to their bare minimum to ensure customizability and reusability. How-

	Off-the-shelf	Paper-Membrane	Pull-chain
Interaction Style	Tangible	Touch	Tangible
Minimal Assembly	Yes	No	No
Seamless Integration (Non-protruding)	No	Yes	Yes
Customisable	No	Yes	Yes

Table 4.1: Strengths and limitations of PaperPulse widget families

ever, we envision more custom designs in the future, such as sliders with non-straight tracks or even circular shapes for dial mechanisms (often called wheels or volvelles in paper craft [26]). The paper-membrane and pull-chain widgets mainly focus on standard controls, such as push buttons, switches, sliders and radio buttons since these components benefit much from customization. However, in the future we hope to integrate paper versions of other input (e.g. bend, pressure sensors) and output components (speakers [128, 135], microphones) in PaperPulse.

4.5 Pulsation: Specifying Sensor Logic By Demonstration

Pulsation allows users to specify logic by demonstrating and recording actions directly in the context of the visual design elements. This preserves the WYSIWYG paradigm, which users are familiar with from graphical software tools. Demonstrating actions in a graphical user interface, however, is limited to actions that can be defined through

the interface of the tool. For example, demonstrating multiple actions that need to be executed simultaneously is impractical using a regular mouse and keyboard. Similarly, specifying a set of actions that can be performed in any order, requires demonstrating all possible permutations. To address these challenges, and provide a higher ceiling than is possible with demonstration alone, Pulsation augments widgets and the demonstrated actions with dialogs that allow fine-tuning of specific properties.

At the same time, demonstrating actions in the context of visual design elements calibrates the state of the input widget to real world values that are present in the visual design. This makes it possible, for example, to gauge a slider by demonstration, or choose which state of a switch is high or low.

To define the behaviour of electronically augmented paper designs, the Pulsation logic recorder supports *if–then* as well as *map–to* rules as shown in Figure 4.1a. For if–then rules, a set of recorded actions (*output set*) is executed when a set of recorded conditions (*input set*) has been met. For map–to rules, parameters of *input set* (e.g. the number of fulfilled actions in the set) are continuously mapped to parameters of the *output set* (e.g. speed with which the set of actions are executed repeatedly). Both *if–then* and *map–to* rules thus relate an *input* event to an *output* event.

4.6 Architecture and Implementation

The design tool supported by PaperPulse, the Pulsation logic and interpreter are implemented in .NET/C#. This section describes the ar-

chitecture and algorithms underlying the PaperPulse system.

4.6.1 Pulsation Interpreter

The Pulsation interpreter can execute recorded if–then and map–to rules in our test and debug environment as well as on microcontrollers. The implementation is consistent with .Net Micro Framework specifications to ensure its portability to microcontrollers, such as Netduino and Threadneedle. As such, the results observed in the test and debug environment of PaperPulse are always consistent with the output from the microcontroller.

To get the recorded logic onto these microcontrollers, we generate code with .NET CodeDOM that re-instantiates all objects needed for the specified Pulsation logic. Once the microcontroller starts, it runs the generated code and thus initializes all logic. Afterwards, the microcontroller runs the Pulsation interpreter every CPU cycle. The Pulsation interpreter keeps track of timing information and states of widgets over different cycles to ensure that the output is always correct and independent of the speed of the microcontroller. The current version of the Pulsation interpreter requires a least 26 kilobytes of memory.

Pulsation achieves a modular design that is reusable and extensible by abstracting: (1) Widgets according to their input or output type to make the system sensor-agnostic (e.g. whether an off-the-shelf slider, paper-membrane slider or pull-chain slider is used, is irrelevant for Pulsation). (2) Connection pins to support different microcontroller platforms, such as Netduino and Threadneedle. (3) Actions and conditions as discussed in sections *Input Sets* and *Output Sets*.

4.6.2 Filtering Signal Noise

In contrast to the behaviour of widgets inside the design tool, their physical counterparts are subject to noise which might lead to undesired oscillations. PaperPulse mitigates this problem by smoothing analog input signals. When analog signals are discretised (e.g. for pull-chain radio buttons), hysteresis, or double thresholding is used.

4.6.3 Generating Electronic Circuits

Similar to Midas [129], PaperPulse employs an auto-routing algorithm to generate conductive traces that connect the pins exposed by widgets to the pins of a microcontroller. We implemented a variation of the A* algorithm [55], in which traces can make junctions with other traces that connect to the same pin. Our routing algorithm avoids other conductive traces as well as the instructions that are printed. When the circuit is non-planar, the algorithm leaves space for a zero-ohm SMD resistor, which serves as a bridge.

Control pins of widgets can often be connected to multiple pins on a microcontroller. This depends on the input or output signal that is required. For example, the anode of an LED can be connected to any PWM pin. However, if binary output suffices, a digital pin can be used. Our routing algorithm takes this into account and first uses the specified logic to assign a set of valid control pins to every widget. The algorithm then selects those pins that maximize the number of widgets that can be connected given the limited set of pins on the microcontroller. Finally, it favours those pins which, when routed, have the lowest number of intersections with other traces.

4.6.4 Generating Printable Pages

Although our design tool gives users the impression that the final design consist of a single sheet of paper, every widget adds content to multiple sheets (see section *Design Challenges*). These sheets consist of conductive traces, visual design elements, and instructions for attaching components, or cutting, folding, and gluing of paper. Each type of instruction has a unique style, such as dotted lines for cutting, dashed lines for folding, and hatched regions for gluing.

Although every design consist of three sheets of paper, some sheets (i.e. the top layer) also have information present on the back of the paper while others require conductive as well as non-conductive information on the same page. Therefore, five PDF files are generated for every design using the PDFSharp library[7]. The tutorial assists users to print these files using the conductive inkjet printer, or a regular colour printer for non-conductive elements. Conductive traces are rendered using vector graphics to preserve the quality and maximize its conductivity. When content is printed on the back of a sheet, PaperPulse automatically flips it to ensure correct alignment. Regions of different layers that have to make contact to ensure electrical connectivity are enlarged to compensate for possible misalignments by the printer or user (e.g Z1 and Z2 in Figure 4.6).

[7] http://pdfsharp.com

4.7 Evaluation

To gauge the usability and utility of PaperPulse, we conducted a preliminary first-use study with four participants. All participants had a design background (1 multimedia, 1 graphical, and 2 product design). Two participants had no prior experience in programming or electronics. The other two participants had some limited experience with Arduino and programming. Every session lasted for 2.5–3 hours. A video introduced the participants to the basic options of PaperPulse. Next, a video tutorial for designing and fabricating the diet card, shown in Figure 4.2, was provided. For the first task, participants were instructed to replicate this diet card using PaperPulse. For the second task, participants had to design and conceive their own ideas in PaperPulse, and reported on their experience with the system through a questionnaire and interview.

All participants were able to design and assemble the diet card in less than 45 minutes. Participants perceived the process of assembling the design enjoyable and were satisfied with the end result and reported that the outcome met their expectations. One designer said he was "pleasantly surprised and the whole fabrication process was like magic".

After finishing the diet card, participants were enthusiastic to make their own design and logic in PaperPulse. Two participants had very concrete ideas: one designed an interactive placemat for restaurants, and the other designed interactive city maps as shown in Figure 4.9: one to filter through points of interest, and another to enable voting for specific neighbourhoods (similar to [153]). The other two partici-

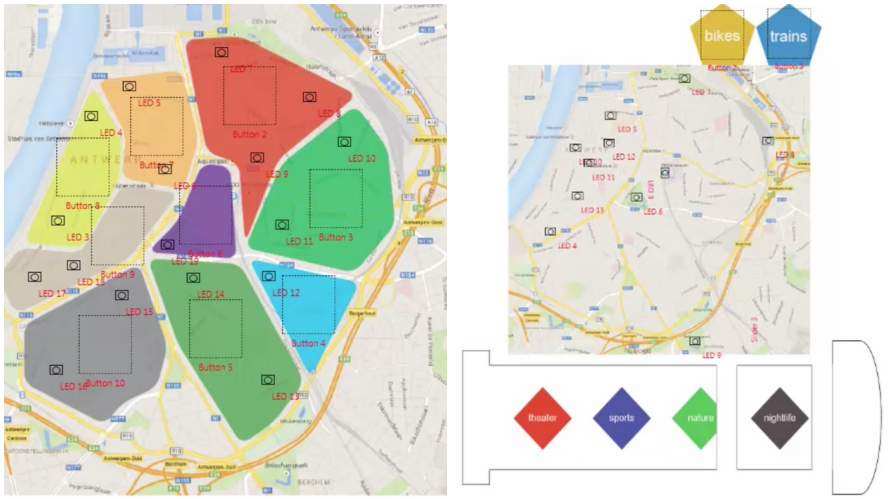


Figure 4.9: Designs made by a participant. (a) A voting meter for neighbourhoods. (b) A tourist information map.

pants had more abstract ideas (e.g. pressing multiple buttons to make LEDs blink, and specify beeping patterns played by a buzzer) and explored these using PaperPulse. During logic specification, all participants used the simulator regularly, to check if the rules they added behaved as expected. Since rules used by participants were quite simple, errors were detected immediately. We expect users to take advantage of the 'Debug View' for more complex rules. All participants could successfully define and fine-tune the interactive behaviour of their designs with Pulsation.

According to the questionnaire and interview, participants felt that PaperPulse supports a wide variety of widgets which could even foster new design ideas. One participant suggested additional widgets that can be supported in the future, such as 2D touch pads and stepper

motors. During the limited exposure to Pulsation, participants found map–to rules harder to understand compared to if–then rules. However, everyone recognized that the derived parameters supported by map–to rules are very useful and provide a lot of flexibility.

The two participants who had experience with the Arduino platform reported that they would be able to make the diet card using other tools, such as breadboards and copper tape. However, they noted that this would require more time and skill and the result would probably not be as visually pleasing as with PaperPulse.

Participants also identified several areas for improvement. Firstly, participants found it hard to get a grasp on the different options available in Pulsation. As suggested by two participants, more comprehensive video tutorials would help give a better idea of how the options can be used in different scenarios. Secondly, participants preferred more visual instructions (e.g. images or videos) during the assembly phase.

4.8 Beyond Paper: Smart Clothing and Home Interfaces

This chapter has extensively discussed the integration of interactive electronics onto paper interfaces. During the course of my research, I have also investigated how we can facilitate the placement of interactions onto other post-WIMP interfaces such as smart clothing and home interfaces. in this section, I briefly discuss our work on these two application areas. A thorough description of these projects is beyond

the scope of this dissertation. For further details, readers are referred to the following publications:

1. **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Enabling eyes-free interactions on jacket buttons. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI EA '14, pages 1549–1554. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2474-8. doi: 10.1145/2559206.2581155
2. **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Inconspicuous interactions on jacket buttons. In *Proceedings of the 2014 CHI Conference Workshop on Inconspicuous Interactions*, CHI EA '14. ACM, New York, NY, USA, 2014
3. **KASHYAP TODI**, KRIS LUYTEN, and ANDREW VANDE MOERE. Making smart homes personal: Fabrication and customisation of home interfaces. In *Proceedings of the CHI '15 Workshop on Smart for Life: Designing Smart Home Technologies that Evolve with Users*, CHI EA '15. 2015
4. JELCO ADAMCZYK, **KASHYAP TODI (ADVISOR)**, and KRIS LUYTEN (PROMOTER). Easyhome: (re)-designing your home interfaces. Bachelor Thesis. 2016
5. STEVEN PEETERS, **KASHYAP TODI (ADVISOR)**, and KRIS LUYTEN (PROMOTER). The home logging toolkit. Bachelor Thesis. 2017

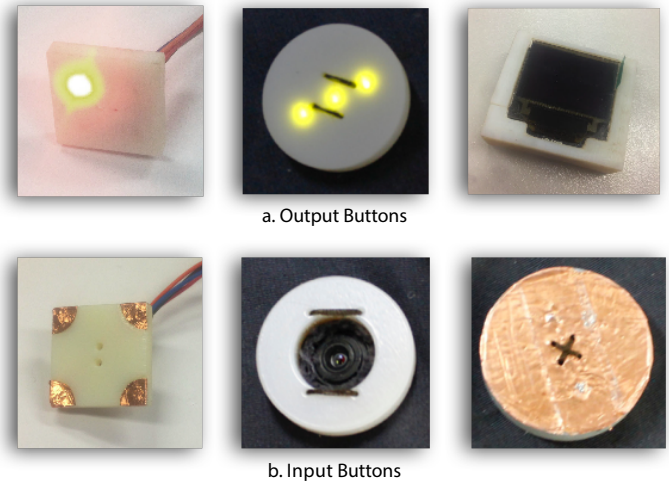


Figure 4.10: Prototyping interactive buttons. (a) A jacket with buttons of suitable size. (b) Some working prototypes: 1. Four-way control 2. Status LEDs 3. OLED Display 4. Piezo-electric Speaker.

4.8.1 Placement of Interactive Electronics on Smart Clothing

In Suit Up! [143], we presented a new interaction space for smart clothing. We augmented ordinary buttons with electronic components, to integrate them into outdoor clothing such as suits and jackets. These interactive buttons, or *iButtons*, allow users to perform tasks using *inconspicuous gestures*—subtle actions which are not easily perceived by others. Different types of buttons serve dedicated functions, and appropriate placement of these buttons make them easily accessible, without requiring visual contact. The buttons are modular in nature; by linking multiple buttons, users can specify customised workflows for certain tasks. We implemented a family of interactive buttons:

Input Buttons: Push button; radial dial; four-way control (D-pad); capacitive touch and proximity sensor; microphone.
Output Buttons: Camera; OLED display; 7-segment display; speaker; dual-state LEDs.
Some of these prototypes are illustrated in Figure 4.10.

4.8.2 Placement on Home Interfaces

Home interfaces consist of a large number of interconnected devices and controls, and can often be quite complex. Further, requirements from a home interface can change over time—with the evolving needs of the user, or as and when different users inhabit the home. This makes placement of interface elements in a home environment challenging and non-trivial. To this end, we propose the concept of an evolving home, which can change over time, and be adapted by the users. Additionally, we developed two end-user tools to place and specify interactivity within the home, and to log interactions with home interfaces.

The Evolving Smart Home

In 'Making Smart Homes Personal' [144], we proposed a framework for the personal home, where the occupant actively contributes to designing and customising the surrounding interfaces. Our proposed model, the Design–Deploy–Dispose (DDD) cycle, is an adaptation of the well-known Design–Implement–Analyse (DIA) cycle, often used for iterative design of interfaces [24]. The DDD cycle, as shown in Figure 4.11, consists of the following phases:

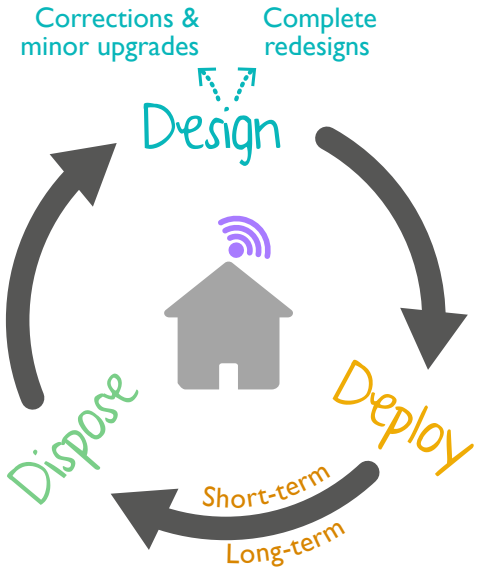


Figure 4.11: The Design–Deploy–Dispose Cycle as a framework for the evolving smart home. Users design and deploy smart interfaces in their homes, and dispose them once their needs evolve, starting the cycle again.

1. **The Design Phase:** End-user design tools enable users to design, customise, and personalise household interfaces.
2. **The Deploy Phase:** Using accessible home fabrication techniques, users assemble the interface, and deploy them in their household.
3. **The Dispose Phase:** As and when users' needs evolve, or the end result is unsatisfactory, the low cost of hardware enables them to recycle or dispose parts, and repeat the cycle.

End-User Configuration of Home Interfaces

PaperPulse investigated an integrated workflow that could enable non-expert users to place interactive electronics on paper interfaces. To enable the continuous evolution and customisations of home interfaces, in 'EasyHome' [2], we extended this to also address the placement of household interfaces, and specify how they interact with each other. Figure 4.12 illustrates such a workflow, where a user starts by overlaying the blueprint of their home with different household widgets, such as buttons and lamps. After configuring the widget, to specify the hardware being used, connections between widgets can be created visually, and the rule editor can be used to specify behaviour.

End-User Logging of Interactions

In chapter 3, Familiariser logged a user's interaction history to automatically adapt user interface layouts to each user. It can be beneficial to apply similar approaches to homes, which also consist of several interconnected interface elements. However, detecting user interactions with ordinary household interfaces is non-trivial. While smart home appliances are capable of self-sensing and logging user actions, the same does not apply to traditional interface elements such as light switches, door handles, blinds, among others. This makes it challenging to model a user's interaction history. To address this, we investigated attaching low-cost off-the-shelf sensors to home interfaces. We implemented a tool, and an end-to-end workflow (Figure 4.13), that guided users towards placing sensors onto home interfaces, and calibrating them, thus enabling sensing of all interactions with such in-

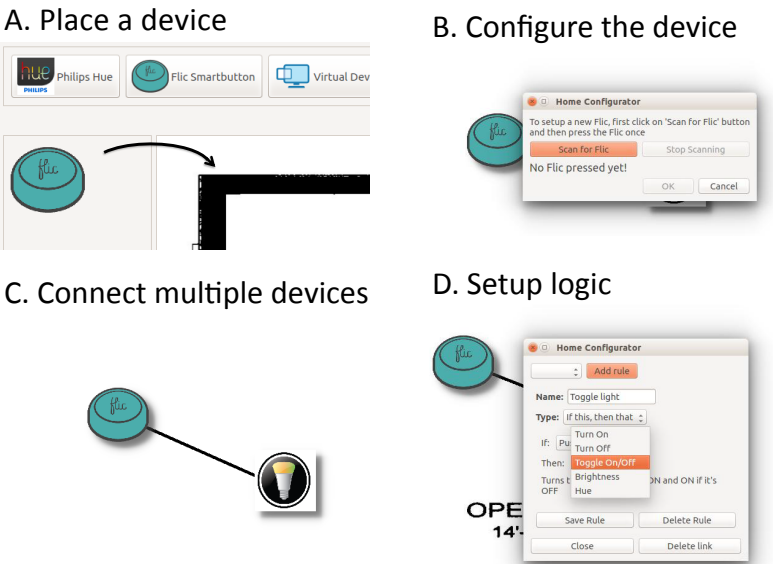


Figure 4.12: Workflow for interactively placing home interfaces. A) The user places a household widget by overlaying it onto the home layout diagram on the canvas. B) The widget hardware is configured. C) The user specifies interconnections between multiple widgets. D) Logic is specified using the rule editor.

terfaces. As a result, different physical interfaces are augmented with logging capabilities. A user's interaction history can be now recorded, and be viewed in the system. By constructing a detailed history of user interactions, we can develop methods and techniques to automatically suggest changes and adaptations in the home interface layout, to improve usability of such interfaces.

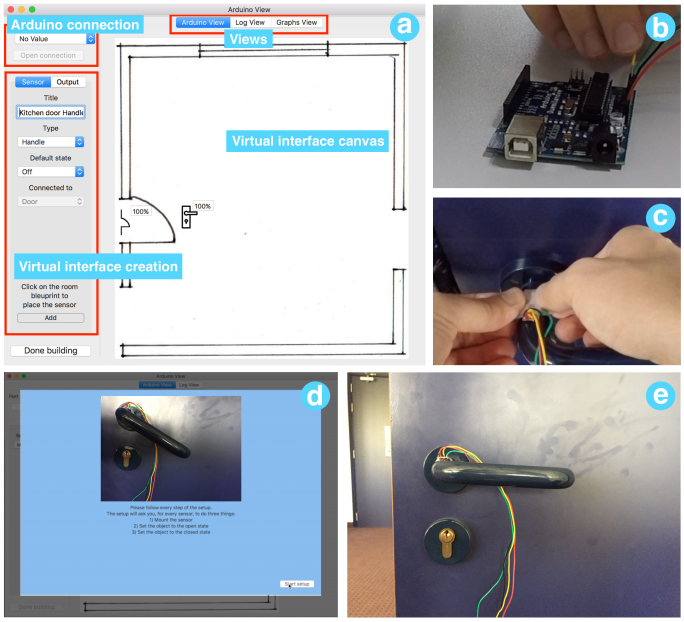


Figure 4.13: Workflow for placing sensors on home interfaces, to log user interactions. (a) The user specifies the home interfaces on a layout diagram of the room. (b and c) The user attaches a sensor to the home interface, and connects it to an Arduino microcontroller. (d) The sensor is calibrated using on-screen instructions. (e) The final interface augmented with interaction logging capabilities.

4.9 Discussion

4.9.1 Summary

This chapter discussed the placement of interactive electronic components on post-WIMP interfaces. It presented *PaperPulse*, a design and fabrication approach that allows end-users to enrich traditional visual designs on paper by placing interactive electronic elements onto them.

In order to do so, PaperPulse contributes a design tool, three families of interactive widgets, and a logic recording and demonstration technique *Pulsation*. PaperPulse supports the whole process from design and specification of interactive paper to fabrication and assembly. An informal evaluation with non-expert users indicated that our approach is viable. Participants were pleased with the resulting standalone paper artefact. They were in particular enthusiastic about the possibilities PaperPulse offers, i.e. creating interactive paper designs, that were unavailable for them before. In the last part of this chapter, I also briefly discussed our work on facilitating the placement interactive elements on other post-WIMP interfaces such as smart clothing and home interfaces.

4.9.2 Revisiting the Research Question

Post-WIMP user interfaces open up a new range of interaction possibilities to communicate with computers, which were previously not possible with GUIs. For instance, by embedding interactivity into physical mediums such as paper, we can create dedicated user interfaces for certain tasks, and move one step closer towards the vision of ubiquitous computing. However, the technical complexities also open up new challenges in placing interactive elements on such interfaces. In our quest to facilitate the placement of interactive elements on post-WIMP interfaces, the research question for this chapter, as stated in the introduction, is:

How can we facilitate non-experts in placing interactive electronic elements on commonly used physical mediums by obviating the needs for pro-

gramming and electronics skills?

To this end, this chapter makes the following technical contributions:

1. *A design tool* to place electronics onto paper designs, and specify, test, and debug logic between these components. When fabricating, our tool assists by automatically generating circuits, layers, pages and instructions to help assemble the final paper artefact.
2. *Pulsation*, a demonstration technique to specify interaction logic between basic electronic sensors. The Pulsation interpreter runs in a simulator integrated into the design tool and on the supported microcontrollers.
3. *Three families of interactive widgets* to support a diverse range of interactive electronic elements. Each one of them consist of multiple standard controls, such as push buttons, switches, sliders and radio buttons for an overall number of 20 different interactive components.

Our evaluation assesses the usability and utility of PaperPulse for non-experts.

By enabling non-experts to place interactive elements onto paper, and thus create their own interfaces from scratch, this chapter opens up the possibilities for highly customised post-WIMP user interfaces. Going beyond paper, chapter also discusses extensions of the concept to facilitate placement on personal post-WIMP UIs, such as smart clothing and household interfaces.

4.9.3 Principles for Facilitating Placement of Electronics

The key design principles derived from this work, towards the goal of facilitating placement of electronic elements on post-WIMP UIs are:

1. **Eliminate need for circuit design** by automatically generating valid circuits based on placed elements.
2. **Support intuitive methods for specifying logic** by using if–then and map–to styled rules on a WYSIWYG interface.
3. **Assist in placing components** by creating specialised hardware widgets and providing detailed assembly guidelines.
4. **Enable iterative design exploration and improvement** by providing a simulator for testing, or by enabling continuous logging, and use-time modifications and customisations.

4.9.4 Limitations and Next Steps

Technically, our implementation of PaperPulse has three main limitations. Firstly, Pulsation is not a general programming language (i.e. Turing complete) that supports arbitrary data structures, functions and variables. We found one could use workarounds (e.g. using the state of an LED as boolean variable) but these come at the expense of simplicity. Secondly, although some widgets draw inspiration from pop-up mechanisms, more extensive pop-up and origami techniques can be integrated in the future to enable non-flat designs. Although the visual design and dimensions of paper-membrane and pull-chain

widgets can be customized, their overall shape (e.g. shape of handle) is fixed. We envision a widget editor in the future. Finally, the current version of PaperPulse does not optimize usage of electronic components. Every widget needs to be exclusively connected to one digital or analog pin on the microcontroller. Future implementations could optimize this by supporting multiplexing strategies or by sharing pins among output widgets that are in the same state at all times. For some very simple designs, widgets could be operated using only a battery, eliminating the microcontroller. While PaperPulse is a rich design tool that facilitates non-experts to place interactive elements on paper interfaces, it does not support modification of the interface *after* it has been constructed. For customised and personalised interfaces, user needs and requirements can change over time. Our work on smart clothing and home interfaces has explored facilitating adaptation of personalised interfaces based on needs and requirements of users.

In contrast to the specialised interactions presented in this chapter, standard input controls enable users to perform elementary tasks such as text and number entry, and navigation. For GUI interfaces, the mouse and keyboard provide users with efficient input capabilities for these tasks. Post-WIMP interfaces, in contrast, support different sensing mechanisms, interaction techniques, and modalities. The input capabilities are limited to the constraints of the interface. In the next chapter, I discuss facilitating the placement of standard input controls on post-WIMP interfaces.

4.10 Acknowledgements

Firstly, I'd like to thank my collaborators, Raf Ramakers and Kris Luyten, for their significant contribution towards PaperPulse. Jo Vermeulen engaged us in many useful discussions, Tom De Weyer and Johannes Taelman provided technical advice, Karel Robert helped with illustrations used in the designs, and Johannes Schöning provided some early feedback on this work. I thank them for helping this project reach fruition. Finally, I also thank study participants for their time.

Chapter 5

Facilitating Placement of Input Controls Across Interfaces

In the previous chapter, I discussed the placement of electronic components, to create special-purpose post-WIMP interfaces. The interaction mechanisms in each interface were highly-customised, and manually specified by the users. In contrast to task-specific interactions, standard input controls enable users to perform elementary tasks such as text and number entry, and navigation. Typically, mouse and keyboard devices enable users to provide such input to GUI interfaces. To address constraints of post-WIMP interfaces, or to take full advantage of their capabilities, novel input techniques have been extensively researched. In this chapter, I address the challenge of placing standard input controls on post-WIMP interfaces. I present *BinPut* as a universal input technique for post-WIMP interfaces. The technique facilitates the placement of input controls across a wide range of devices and modalities, and for different types of input, such as text or number

entry, selection, and navigation. BinPut can be directly applied across a diverse range of scenarios, while maintaining consistent behaviour and usability, and supporting transfer of learned skills.

5.1 Introduction

A long-term goal of the post-WIMP and ubiquitous computing community has been to seamlessly weave interactivity into the fabric of our daily lives by augmenting a wide range of surfaces and spaces with computational power. Unlike traditional GUIs, post-WIMP interfaces and technologies open up possibilities for computing devices tailored towards specific purposes, and designed to handle specific tasks. In the previous chapter, I discussed some such post-WIMP interfaces, in the form of interactive paper, wearables, and home interfaces. I investigated the facilitation of placing specialised electronic widgets and components onto post-WIMP interfaces. These widgets enabled customised workflows and interactions, and were targeted towards specific tasks. However, standard input controls for tasks such as text entry, item selection, or navigation were not explicitly considered.

Input devices such as standard keyboards tend to have a large footprint, and specific device requirements. Placing them onto post-WIMP interfaces is often either tedious or impractical. It can be beneficial to minimise the footprint of input controls, yet enable standard input tasks on new interfaces. A typical approach for integrating such input into new devices and modalities is to develop custom input techniques specifically for each device. Switching between devices, therefore, requires us to learn different input techniques, and continually change

or adapt our input strategy accordingly. Additionally, the complexity of specialised input techniques makings it hard for users to place them on interfaces.

5.1.1 Research Question

Fallman states that ubiquitous computing moves the focus from tools tailored to a specific task to designing consistently "good user experiences" [38]. This chapter is, therefore, motivated by the potential for a low-threshold and consistent input technique, which can be applied to a wide range of devices and modalities, and for various types of input tasks. Further, to aid in placement of input controls onto a variety of post-WIMP UIs, we aim to reduce the footprint required for the controls, and to have a technique that can directly scale across different interfaces, with no or minimum modification. This inspires the following research question:

How can we facilitate the placement of standard input controls on post-WIMP interfaces while maximising consistency across interfaces and reducing re-learning of the input technique?

A technique that is consistent across devices and modalities, and has minimal technical requirements, offers the advantage that new interfaces can directly adopt it, and present users with a consistent technique, thus reducing adaptation and re-learning requirements. To this end, we present *BinPut*, an input technique that is both device-independent and input type-independent.

5.1.2 BinPut: An Input Technique for Post-WIMP Interfaces

BinPut formulates input tasks as a targeted search problem. Using the binary search mechanism [20], users recursively traverse any ordered input search space to reach target elements. Examples of such ordered input sets include numerical values, alphabets, sorted lists, and continuous positions (e.g. sliders and cursors). We adapt binary search, an efficient and scalable search algorithm, to enable universal input with minimal device requirements. With BinPut, input tasks only require two navigational commands—*increment* and *decrement*—and two additional commands to *confirm* and *undo*. The technique scales to multi-dimensional input, such as pointing, as well. For every additional input dimension, two additional navigational commands are required. Since the technique can be applied to any ordered input set, it is *type-independent*. BinPut also scales across devices and modalities since it requires a minimal number of input commands, and can thus be considered *device-independent*. The technique finds applications in ubiquitous interfaces, emerging and novel devices, do-it-yourself (DIY) interfaces, and in scenarios where accessible input is required. BinPut makes it possible to place input controls into UIs where it might otherwise be infeasible or complex—it minimises the required footprint, and is easy to implement, requiring minimal computational power.

In this chapter, I describe the BinPut technique in detail, discuss type- and device-independence aspects, and also highlight opportunities for customising the technique for specific scenarios. We present a theoretical evaluation to illustrate the feasibility and benefits of our ap-

proach. Our user study evaluates whether the BinPut technique supports the transfer of learning between devices, for different types of input.

5.2 Background

There is a long-spanning history of input-related research. Our work is closely related to other efforts to enable input for post-desktop and ubiquitous interfaces, and also to research related to device-independent input. The following discussions do not aim to provide an exhaustive list of emerging input techniques, but summarise key aspects of these two areas of research, and distinguish our work from previous research.

5.2.1 Post-WIMP Input Techniques

Previous research efforts have have focussed on enabling text-entry on post-desktop and ubiquitous devices with varying input capabilities, such as (multi-)touch input, gestural, midair, and gaze interactions. Gestural keyboards [79] helped overcome drawbacks such as lack of haptic feedback, and enabled one-handed interactions with touch-sensitive screens. For smaller surfaces such as smartwatches and wearables, works such as [110, 27, 51] have proposed novel text-entry techniques that are tailored towards target devices. For midair interactions, handwriting recognition [3] and gestural interactions [96] have been developed to deal with problems such as lack of hand-support and haptic feedback, or jittery input and unstable motions.

Each of these text-entry techniques has been tailored towards the given technology, addressing problems inherent to them. In a similar vein, pointing and selection techniques have also been implemented for post-desktop devices. Direct touch input for targeting on touch-sensitive surfaces provides obvious benefits. However, as the input area shrinks, disadvantages of these direct manipulation techniques, such as occlusion and the fat-finger problem, have become apparent. Works such as [22, 114] have improved targeting and selection on small surfaces by moving the interactions to surfaces around the device. Pointing and selection in midair has been investigated as well. AirPointing [29] studies target acquisition without visual feedback, and [154] describe a technique for freehand targeting from a distance, for large displays. [95] enables selection of hyperlinks using gaze interactions.

It is noticeable that each of these works has provided tailored solutions that take advantage of capabilities offered by new technologies, and also attempt to overcome hurdles introduced by them, for enabling different types of input. These research works aim to best support a given device or modality, for certain types of input. The input technique, therefore, needs to be customised and adapted for specific scenarios are conditions. In contrast, BinPut aims to provide a universal solution, that can be applied to a diverse range of input types, and adopted by different devices and modalities.

5.2.2 Device-Independent Input and Input with Few Keys

Previous works, such as [65], have presented device-independent input techniques. This is typically done by reducing the number of keystrokes, gestures, or input events, required to support input tasks. EdgeWrite [160] is one such instance that promotes cross-device text entry, and it has found uses in several domains such as in-car text-entry [47]. The technique uses four directional moves to create patterns resembling Roman character glyphs, enabling text-entry on devices capable of recognizing four distinct input strokes or gestures. However, the technique is limited to text-entry, and is designed specifically for the A—Z character set. Directly applying it to a different language or character set is non-trivial. Like EdgeWrite, Dasher [155] also offers a device-agnostic text-entry mechanism. Dasher is designed such that it can provide text-entry in situations where a full-scale keyboard is not feasible, and it claims to work for any language, hence providing more flexibility. It has also been adapted for different modalities such as gaze input [148]. There have been works such as [93] that also highlight benefits of interactions with a small number of keys. However, since they use specific encodings, they can not be easily ported across input sets, and require users to learn and remember the encodings.

As evident, the above techniques are limited to one type of input—text-entry, and often to a specific character set. To our knowledge, the only other technique that is agnostic to both input modality and input type is linear or sequential search [14], where input is selected by seri-

ally traversing all elements. However, the linear search mechanism is not performant, and does not scale well to large input sets, and hence is used sparingly.

The binary search technique, like linear, also has the potential of enabling device-independent input that works across different input types and tasks. Previously, [86] studied the application of binary search for list selection tasks, and results from their study favoured the binary search mechanism. In our work, we adapt binary search, and present mechanisms through which the technique can be applied an extensive number of scenarios, for different input types and tasks, and directly adoptable by various input devices and modalities.

5.3 BinPut: Adapting Binary Search for Input

Binary search, also known as *logarithmic search*, requires a maximum of $log_2(n-1)$ comparisons to locate an element within a sorted array containing n elements. This implies that as the size of the search space grows exponentially, the number of comparisons needed to locate any element increases linearly. Given no additional information about the search space or the target element, binary search offers an optimal search technique. In our work, we adapt the binary search mechanism to present a flexible input technique that can be applied to any ordered input set.

5.3.1 Walkthrough: An Input Task

In *BinPut*, the input task is essentially reduced to targeted search. To find a target element, within an ordered input set, the following steps are performed:

1. Initially, the middle element of the input space is presented as a *cue* to the user.
2. The user mentally compares the cue element to the target element, and makes a move in the required direction, to get closer to the target.
3. The cue element updates accordingly, and gives the user feedback about the update state.
4. The user repeats navigational moves, to recursively traverse the ordered set, and reduce the input space.
5. Once the cue and the target elements match, the user selects the input.

5.3.2 Input Commands

BinPut provides users with four essential input commands (or *moves*):

1. **Increment**: Performed when the desired input element appears after (proceeds) the current cue (middle element).
2. **Decrement**: Performed when the desired input element appears before (precedes) the current cue.
3. **Confirm**: Performed to select the current cue as the target.
4. **Undo**: Reverts the search space to its previous state.

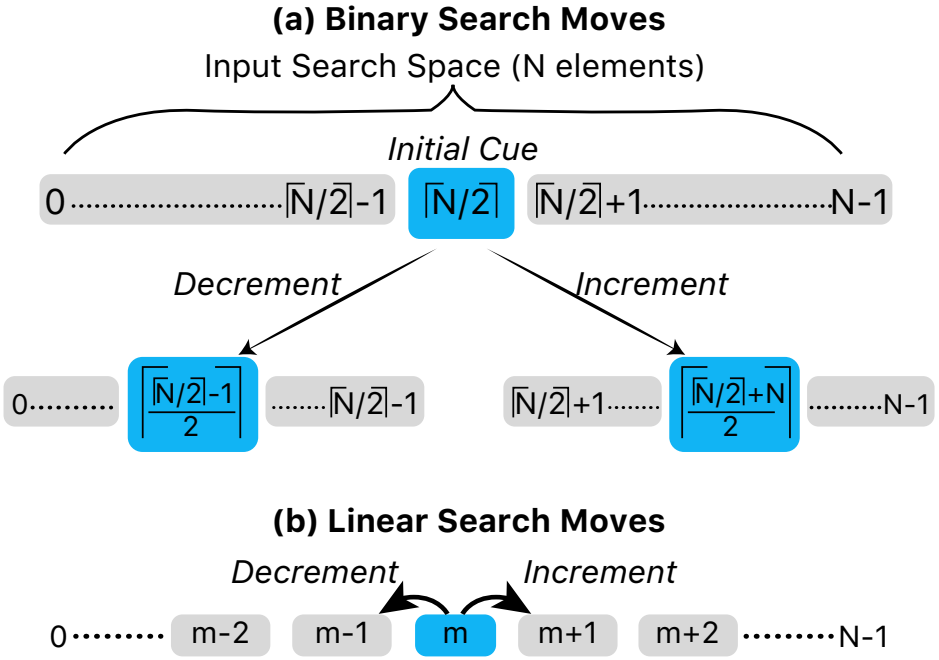


Figure 5.1: Searching the input space. (a) Binary search traversal when starting from the initial input space of N elements. An *increment* or *decrement* move reduces the search space to half the original size. (b) Linear search moves when the search space has <=5 elements.

5.3.3 Searching the Input Space

To locate target elements in the input space, BinPut combines both binary and linear search techniques. A User does not need complete information about the input set, or exact positioning of target elements. Instead, they compare the current cue (middle element) to the target element, and decide whether an increment or decrement move is required.

Traversing with Binary Search

When the input space consists of more than 5 elements, the binary search mechanism is applied to navigate the search space (Figure 5.1a). To find a target element within the sorted array, a single comparison is made with the current cue (middle element). If this cue element is greater than (proceeds) the target, a *decrement* command limits the consequent space to the lower half. Alternatively, if the cue is lesser than (precedes) the target element, the *increment* command trims the search space to the upper half. By recursively performing these comparison and trimming operations, users can quickly locate any target element within the input space.

Switching to Linear Search

When the search space consists of 5 elements or less, the average number of moves required by both binary and linear search are similar. Thus, BinPut switches to a linear mechanism when the search space reaches this threshold ($<= 5$ elements). The search space is reset to the original (entire) array of elements, and users can increment or decrement to linearly scan through the elements, and find the target (Figure 5.1b). This linear mechanism improves usability, and aids in late recovery from errors, providing a fallback search mechanism in cases where users might make erroneous moves while navigating through the binary tree, thus ending up at the wrong leaf node.

The resultant BinPut algorithms for *Increment* and *Decrement* input commands (moves) is summarised in Figure 5.2.

Variables:
A[N]: Input Space with N elements
mid: Current middle index
min: Minimum index of search space
max: Maximum index of search space

```
Increment(A[N],mid, min, max):
1.  if mid == N-1
2.     return A[mid]    (Stop at last value)
3.  else if (max - min) <= 5
4.     mid = mid + 1    (Linear Increment)
5.     return A[mid]
6.  else
7.     min = mid + 1    (Binary Increment)
8.     mid = ⌈(min + max)/2⌉
9.     return A[mid]
```

(a) *Increment*

```
Decrement(A[N],mid, min, max):
1.  if mid == 0
2.     return A[mid]    (Stop at first value)
3.  else if (max - min) <= 5
4.     mid = mid - 1    (Linear Decrement)
5.     return A[mid]
6.  else
7.     max = mid -1    (Binary Decrement)
8.     mid = ⌈(min + max)/2⌉
9.     return A[mid]
```

(b) *Decrement*

Figure 5.2: The BinPut algorithm for (a) increment and (b) decrement input moves. When the search space has $<= 5$ elements, BinPut switches to a linear search mechanism (Step 3–5).

5.3.4 Undo Mechanism

To implement consistent undo, BinPut maintains a *state history* consisting of a sequential array of BinPut states, corresponding to the input moves performed previously. Each state contains information of the current minimum, maximum, and middle values pertaining to a given increment/decrement move. By maintaining a complete state history, the undo command behaves deterministically, irrespective of the sequence of moves performed.

By applying the above search mechanisms, BinPut enables *type-independent* input, in that it can be directly applied to any type of input set or task, as long as the elements exhibit an inherent ordering. Since the technique requires only four elementary input commands, and a single cue (middle element) as output or feedback, BinPut also claims to be a *device-independent* input technique.

5.4 Type-Independence with BinPut

Since BinPut is applicable to any ordered input set, the above mechanisms provide a basis for adapting BinPut to a range of input types. This includes text and number entry, selection within ordered lists, one-dimension scrolling, and two-(or multi-) dimensional inputs such as pointing and navigation.

5.4.1 Number Entry

By initialising the input space with a desired range of values (minimum and maximum), input of numerical data is enabled. For exam-

ple, by specifying a valid range of 'years', one can directly enter a four-digit birth year without individually entering each digit, or scrolling through a long list of values. Number entry within a restricted (ordinal) set can be supported as well. For instance, intermittent values (e.g. multiples of 10 only) can be entered using BinPut. This can be implemented by either specifying a rule (or function), or by providing a precise set of accessible input values.

5.4.2 Text Entry

BinPut offers a viable text entry technique when the input or output device capabilities are limited. Unlike other text entry methods, BinPut is agnostic to the language and character set, making it widely adoptable. The only exceptions are alphabet or character sets that do not exhibit inherent ordering. In addition to character-wise text entry, BinPut can also be applied to whole-word entry, and even phrase-level text entry. Figure 5.3 illustrates the search tree for the English alphabets (A–Z), where any character can be reached with a maximum of five BinPut moves.

5.4.3 List Selection

Selection of elements from long lists is another excellent candidate for BinPut. This is applicable when lists can be sorted according to an identifiable criteria, such as alphabetically, sequentially, or chronologically. To implement list selection, BinPut simply requires the specification of list elements, and the sorting criteria to be applied.

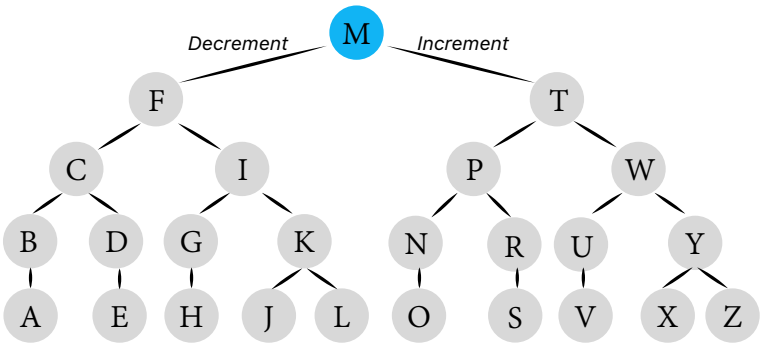


Figure 5.3: The binary search tree for Roman (A to Z) characters. A maximum of 5 moves is required to reach any character.

List selection using BinPut has several use cases. For example, street names, cities, and countries, can be entered into navigation systems using a small number of steps. Browsing through contact lists on a smartwatch, selecting from a large list of movies using a simple remote control or gestures, or finding a photo, in a collection sorted chronologically, are other exemplary scenarios.

5.4.4 One-Dimensional Scrolling

BinPut can also be applied to other common forms of input, such as one-dimensional scrolling and navigation. For instance, a media player can implement BinPut for fast and accurate navigation to any particular point on a timeline. (Figure 5.4) illustrates this with an implementation of BinPut for a DIY paper remote control, fabricated through conductive inkjet printing and capacitive touch input. BinPut can be adapted to work from any arbitrary starting position (scroll position). Only the current position in the scrollbar or timeline, and

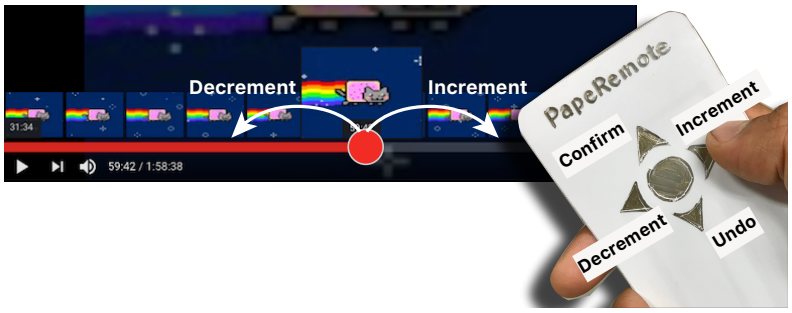


Figure 5.4: BinPut can be used to quickly and accurately scroll through a continuous range of values, for example, to navigate to an exact timestamp in a video using a DIY paper remote control made using conductive inkjet printing.

the accessible range (minimum and maximum) of values, are required. When the user initiates a navigation command, the entire range is first divided into two sub-ranges, one on either sides of the current position, and these are then treated as two independent BinPut instances.

Aceituno et al. [1] state that high-resolution mice are often not fully utilised, and addresses the problem of finding the *useful resolution* of such devices. Since BinPut allows users to recursively approach a target, with increasing precision at each step, it bypasses this problem, and in a way, allows users to "dynamically" determine the required resolution. Additionally, while typical scrolling and pointing movements consists of an initial ballistic phase, followed by corrective phases to recover from undershoots and/or overshoots, BinPut circumvents this by supporting precise navigation to a target position.

5.4.5 Multi-dimensional Pointing

BinPut implements multi-dimensional pointing (or navigation) by simultaneously maintaining multiple BinPut instances, one for each dimension. Two navigational input commands are required for every input dimension.

Indirect pointing on 2D-surfaces can be realised using a total of 4 navigational commands (Figure 5.5). Since the two dimensions are independent, the number of moves required to point at any on-screen target is the sum of moves required along each direction. Thus, for a 1920×1080 display (2,073,600 pixels), the maximum number of BinPut moves required is:

$$\begin{aligned} Max_x = MaxBinPutMoves_{1920} &= 11 moves \\ Max_y = MaxBinPutMoves_{1080} &= 10 moves \\ => Max_{xy} = Max_x + Max_y &= 21 moves \end{aligned}$$

where, Max_x and Max_y are maximum moves for x- and y-axis respectively, and Max_{xy} is the total maximum moves for the 2D space.

As targets usually occupy several pixels, the effective number of maximum moves for 2D pointing is generally lesser.

5.5 Device-Independence with BinPut

As previously highlighted, BinPut is a device-independent input technique. To make the technique adoptable by a wide range of devices and modalities, BinPut minimises the input and output device requirements, as elaborated below.

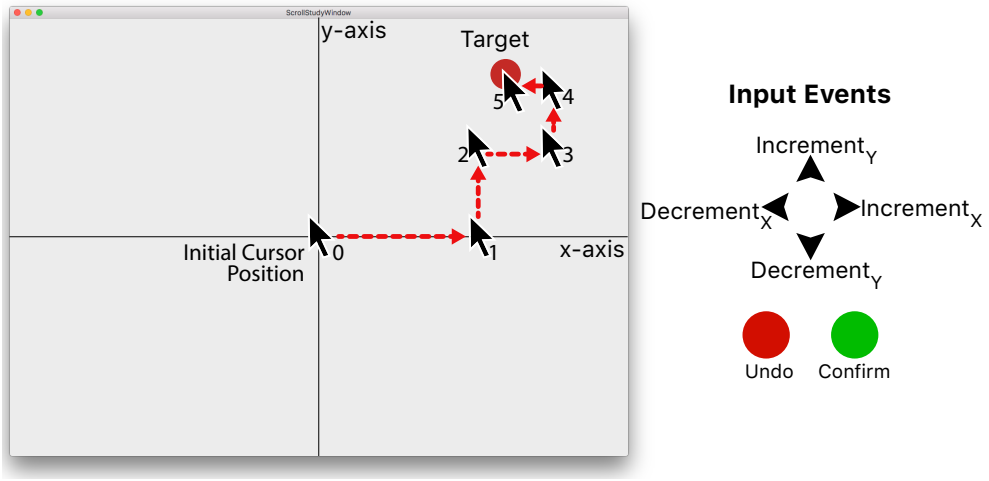


Figure 5.5: Multi-dimensional pointing with BinPut. Each dimension is assigned an independent instance of BinPut, allowing navigation to any point on all dimensions. The figure illustrates a series of 5 moves to manipulate the cursor from the initial cue position to a target. 6 distinct input events are required for 2D pointing.

5.5.1 Input Device Requirements

A complete implementation of BinPut can be realised with just four elementary input commands. Therefore, any input device or modality capable of detecting four distinct events can support BinPut for different input types. For multi-dimensional input tasks, such as 2D pointing, two events (for increment/decrement commands) are required for each dimension. Some common input devices and modalities suitable for BinPut are highlighted below.

Physical or soft keys: This includes input devices such as keyboards, physical keys, and buttons. A large number of devices such as remote controls, microwave ovens, on-steering car controls, often

support only a limited number of input keys.

Touch input: BinPut can be adopted by typical touchscreen devices such as phones, tablets, or smartwatches, among others. Through capacitive and resistive sensing, novel ubiquitous interfaces, such as interactive paper with printed electronics [72] or 3D-printed objects [131], can also enable touch input. For such interfaces, BinPut can provide a reliable input mechanism, even when sensing is of low-accuracy, and only a small number of touch locations (events) are available.

Midair and Gaze Gestures: Midair and gaze input are known to be jittery, inaccurate, and error-prone [100, 53]. This can be attributed to sensing and human-motor limitations. Since BinPut requires only four discrete input moves, it can be effective in such scenarios as well. By employing gestures such as midair swipes, or gaze oscillations (saccades) in fixed directions, input events are distinctly recognised, and used for BinPut moves.

Speech: BinPut can also be used for speech input, for devices where complete speech recognition is computationally infeasible or expensive, or when precise commands are unavailable, such as in noisy environments. This can be especially beneficial for inputs such as list scrolling or pointing. Simple verbal commands for navigation (increment/decrement), confirm, and undo, are sufficient to fully implement the technique.

5.5.2 Output Device Requirements

BinPut has minimal output requirements to provide users with stimulus or feedback. Since comparisons are made with a single cue element, only this needs to be perceivable by users. Neither the entire input space, nor the minimum and maximum values, needs to be provided to the user. However, when rich output is available, additional information about the input space, or the range of values, can improve usability. Thus, BinPut can be successfully implemented with different output devices and modalities, and the level of feedback can be adapted according to device capabilities.

5.5.3 Implementations

During the course of this research, we implemented BinPut on several devices and platforms:

1. **Physical keys on laptops and desktops (MacOS)**: For one-dimensional scrolling, or two-dimensional pointing (Figure 5.5), using a regular keyboard.
2. **Touch input on mobile devices smartwatches (iOS and WatchOS)**: For list selection, text, and number entry (Figure 5.7b) on touchscreens of various sizes.
3. **Midair gestures with the Leap Motion controller**[1]: For free-hand gestural input.
4. **Gaze input with EyeTribe tracker**[2]: For input at a distance, and for accessible input, using discrete eye movements.

[1] www.leapmotion.com
[2] http://theeyetribe.com

5. **Arduino-based input sensors**: For input using sensors connected to an Arduino, such as inkjet-printed capacitive touch (Figure 5.4).

These implementations illustrate the device-agnostic property of BinPut.

5.6 Theoretical Evaluation of BinPut

We investigate the scalability aspects across input types and tasks using a theoretical evaluation of BinpPut. For this, we analyse the number of moves required for input, and the estimated time taken for input tasks.

5.6.1 Number of Input Moves

The number of moves required for an input task is related to the size of the input set. Here, we assess the effect of varying number of elements in the input set, and compare BinPut against two linear search mechanisms:

1. Linear (Start at Middle): Linear search mechanism with the initial cue at the middle of the input space.
2. Linear (Start at First): Linear search mechanism where the initial cue is the first element of the array.

While binary search has a time complexity of $O(log\ n)$, linear techniques are typically $0(n)$. For the above three mechanisms, the maximum number of moves required to select input from a set containing

Input Type	Size of Input Set	Number of Input Moves: BinPut Mean	BinPut Max	Linear (Start at Middle) Mean	Linear (Start at Middle) Max	Linear (Start at First) Mean	Linear (Start at First) Max
English Alphabets	26	3.11	5	6.5	13	12.5	25
Khmer Alphabets	74	4.38	6	18	37	36.5	73
Number Entry 0–99 (e.g. age)	100	4.85	7	24.5	50	49.5	99
Number Entry 0–9999 (e.g. postcode)	10000	11.36	13	2499.5	5000	4999.5	9999
List Selection (with 1000 elements)	1000	8.01	10	249.5	500	499.5	999
1-D Scrolling (2880 pixels Retina Display)	2880	9.58	11	719.5	1440	1339.5	2979
2-D Pointing (2880 × 1800 pixels Retina Display)	5,184,000	18.58	22	1169.5	2340	2339.5	4679

Figure 5.6: Scalability analysis for different input types, with varying set sizes. As the size of the input set increases, the number of moves required by linear techniques rapidly becomes much higher than that needed by BinPut.

N possible inputs can be represented by the following equations:

$$\begin{aligned} MaxMoves_{BinPut} &= \lceil log_2\,(N-1) \rceil \\ MaxMoves_{LinearMiddle} &= \lceil N/2 \rceil \\ MaxMoves_{LinearFirst} &= N-1 \end{aligned} \tag{5.1}$$

Figure 5.6 summarises the mean and maximum number of input commands (moves) required for these three input mechanisms, for different input types and tasks. We cover a range of input tasks, with varying input space sizes, including text and number entry, list selection, scrolling, and pointing.

5.6.2 Input Task Time

In addition to the size of the input space, input performance (task completion time) is also vastly affected by input devices, or modalities, and their accuracy. For a given input task, the task completion time can be estimated as:

$$t_{task} = N_{moves} * (t_{cognition} + t_{motor}) \tag{5.2}$$

where, t_{task} = task time; N_{moves} = number of moves required; and $t_{cognition}$ = cognition time per move, t_{motor} = motor time per move.

Cognition Time

It is evident that the time required for cognitive processing, to decide the next move, plays a key role in overall input time. In contrast to simpler techniques such as linear search or direct pointing, BinPut might suffer from increased cognition time, since users need to repeatedly evaluate the current cue, to decide their next move. This is particularly the case for new users, unfamiliar with the technique. However, since BinPut moves are determinate and predictable, learning and recall could alleviate this issue. Over extended use, it is possible for users to acquire a better understanding of the technique and anticipate the next moves in advance, thus reducing cognition time.

Motor Time

The motor time per move is the amount of time required for a user to make the next input move, after having decided what the move needs to be. This time is dependent on the input devices and modalities, as

well as user abilities. The advantages of minimising the number of moves becomes evident with devices that are slower or less accurate (large t_{motor}). This is typically the case for post-WIMP devices, which are known to support alternate input modalities, or are restricted by technical limitations. Accessible input devices, which address specific user needs, also typically have the limitation that the motor time required for input strokes is often quite large. For example, making a single input move using eye gaze is much more expensive than a button press. Since the cognition time remains constant across devices and modalities, BinPut can also be advantageous in maintaining similar performance across different inputs.

5.7 Evaluation: Device- and Type- Independence of BinPut

Our theoretical evaluation clearly illustrates the benefits of BinPut over linear search techniques. While the technique is easily adoptable across devices and input types, the transfer of learning skills is an important aspect to its success. Thus, we conducted a user study to verify whether users can learn the technique and transfer this to a different device or modality, and a different input type or task.

5.7.1 Study Conditions

The study was divided into two conditions. For each condition, a distinct input device (modality) was used for a distinct type of input task.

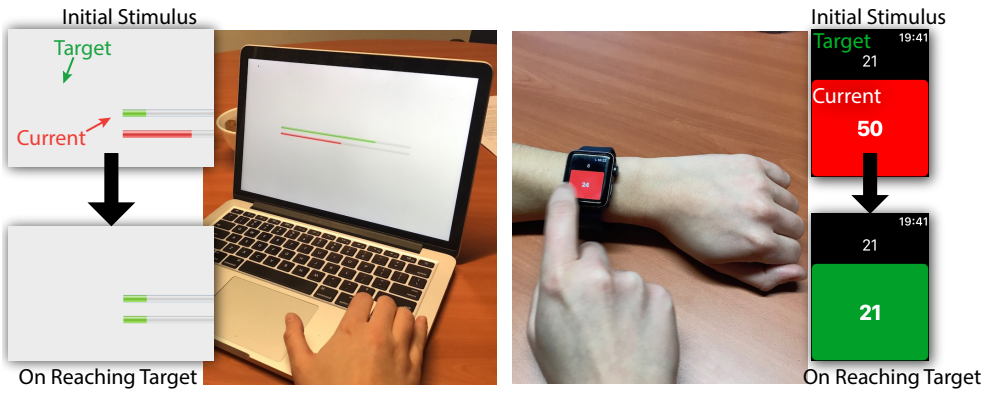
1. **Physical keys for Visual Scrolling:** In the first device–task combination, participants used four physical keys to visually scroll to targets along a one-dimensional horizontal scroll bar, displayed on a screen. The four arrow keys on a standard keyboard were used for input commands. The left and right keys were used to scroll to different positions (search the BinPut tree), the up-arrow was used to confirm an input, and the down to undo previous moves (Figure 5.7a).

2. **Smartwatch Swiping for Number Entry:** In this device–task combination, participants used a smartwatch to enter target numbers. Right and left swipes were used to search for the target numbers, swiping upwards confirmed the input, and swiping downwards allowed undo of previous moves (Figure 5.7b).

To maintain consistency between both scenarios, a total of 100 targets were reachable. That is, the scroll bar was internally divided into 100 distinct positions, and the range for number entry was 0 to 99. Note that for the scroll bar, there was no visual indication of these divisions, and it appeared as continuous on the screen.

5.7.2 Apparatus

For the scrolling task with physical keys, a Macbook Pro laptop running MacOS 10.12 was used. The four arrow keys on the built-in keyboard were used for input. A 15" Retina Display was used to display the stimulus and feedback to participants. All keystrokes were recorded with timestamps, and stored in a CSV file.



(a) Physical Keys + Scrolling (b) Smartwatch Swipes + Number Entry

Figure 5.7: The two conditions in the study. (a) Scrolling using physical keys. (b) Number entry using smartwatch swipes. For each trial, a target stimulus is presented, and the current position (in red) is set to the initial value. On reaching the target value, an indication (in green) is provided.

For the number entry task with swipe gestures, an Apple Watch (42mm, Series 2) running WatchOS 3 was used. A custom WatchOS application was implemented, and run on the device, which displayed the stimulus and feedback. All gestures were recorded with corresponding timestamps, and stored in a CSV file.

5.7.3 Participants

We recruited 20 participants (9 female), aged 23 to 39 (mean 29.7). 7 participants had a background in Computer Science, while the remaining 13 participants had not undertaken an education in Computer Science or Mathematics. All participants used physical keys (keyboard) on a regular basis, but did not use a smartwatch. The participants were

divided into two groups (10 each), and the distribution of educational background and gender in both groups was balanced.

5.7.4 Procedure and Experimental Design

For each participant, the study was divided into two consecutive rounds. Participants performed the two conditions sequentially, with a short break after each round, during which they were asked to complete a short qualitative questionnaire. All participants performed both conditions, resulting in a within-subject study design.

Each round consisted of 120 input trials. Since a different number of optimal BinPut moves are required for different target values, the trials were divided into 12 *sets*, each consisting of 10 trials. We ensured that the total number of moves required for input trials within a set was constant for all sets, making it possible to compare each of them. For the given range of possible target values (0 to 99), the number of navigational moves required to reach all targets ranged from 0 to 7 moves. The distribution of targets within the sets ensured that each set consisted of the entire range of these moves. Exact target values were picked at random from the 0–99 range. Participants were allowed to take a short break after every third set.

To study the learning and skill transfer aspects of the input technique, participants were divided into two groups. Between the two groups, the order in which the above two scenarios was presented to the participants was interchanged. Thus, while the first group performed scrolling tasks with physical keys in *Round 1*, followed by the number entry task with a smartwatch in *Round 2*, the second group

	Condition 1	**Condition 2**
Participant Group 1	Physical Keys + Scrolling	Smartchwatch + Number Entry
Participant Group 2	Smartchwatch + Number Entry	Physical Keys + Scrolling

Table 5.1: Ordering of conditions (device + type) for the two groups of participants, during the two rounds.

performed them in the reverse order (Table 5.1).

Using the above experimental design allowed us to evaluate whether users, after using BinPut for a certain input task with a given device, can apply the input technique to a different scenario, where the type of input (task) and modality are varied.

There are three factors that influence learning and performance (task completion time) in such a set-up:

1. Device Learning (L_{Device})
2. Task Learning (L_{Task})
3. Input Technique (BinPut) Learning (L_{BinPut})

To evaluate whether *Input Technique Learning* transfers from one device and task to another, we can analyse the learning and performance differences between the first and second rounds. To elaborate, the study design enables a comparison of the learning curve and performance of participant group 1, during the first round (*Round 1*), to that of participant group 2, during the second round (*Round 2*). Similarly, we can compare results of participants in group 2 during the first round, with that of participants in group 1 during the second round. Since the device and task are consistent in both these comparisons, the

effect of *Device Learning* and *Task Learning* should not influence the results, and the effect of *Input Technique Learning* is the only factor that would lead to observable differences.

5.7.5 Results

Results from the study indicated that participants were able to transfer their learning of the technique from one device and task, to a different device and task condition. In this section, we elaborate on this this by comparing learning curves for the two rounds, for both conditions, and by comparing average task completion times.

Figure 5.8 shows the learning curves for both conditions, by the order in which they were used by participants. For scrolling with physical keys, participants who were given this condition first (group 1) were able to reach an average set time of 30 seconds at set number 7, that is, after 60 trials. For the same condition, group 2 participants were able to reach this average time of 30 seconds at set number 8, that is, after just 20 trials. For number entry with a smartwatch, group 1 participants reached an average set time of 45 seconds at set number 8, after 70 trials. Group 2 participants reached this average time of 45 seconds at set number 4, after 30 trials. These results indicate that it took fewer trials to reach a consistent level of performance (set time) during the second experienced condition (Round 2), as compared to the first (Round 1). This holds true for both conditions, that is for both device–task combinations.

Thus, while device and task learning occurred in both conditions, we can conclude that participants successfully learnt the input tech-

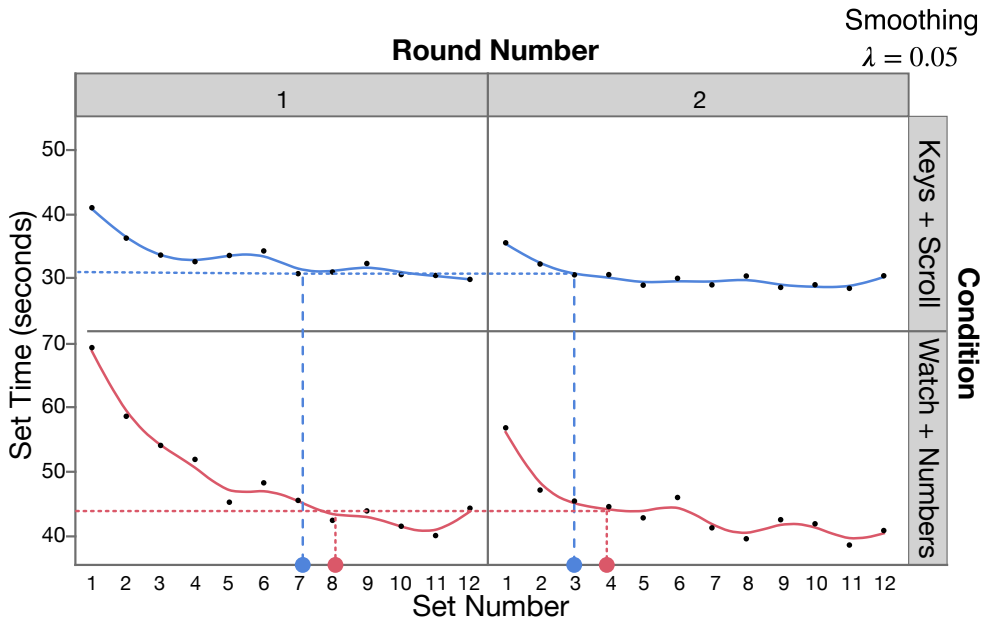


Figure 5.8: Learning curves by round number, for the two conditions. For both conditions, performance become consistent sooner in round 2 compared to round 1, highlighting the transfer of BinPut learning.

nique during the trials in the first round. Since they were able to transfer this to the second round of the experiment, they could reach a stable performance level sooner.

Figure 5.9 shows the average set time for both conditions, for the two groups of participants. Round number had significant effect on the average set time, with consistently higher set times for the first round as compared to the second (Round 1 = 40.25 seconds, Round 2 = 36.47 seconds, $t(18) = 2.19,\ p < 0.05$).
Additionally, condition (device + task type) had significant effect on the set time (Physical keys + Scroll = 30.48 seconds, Watch + Number

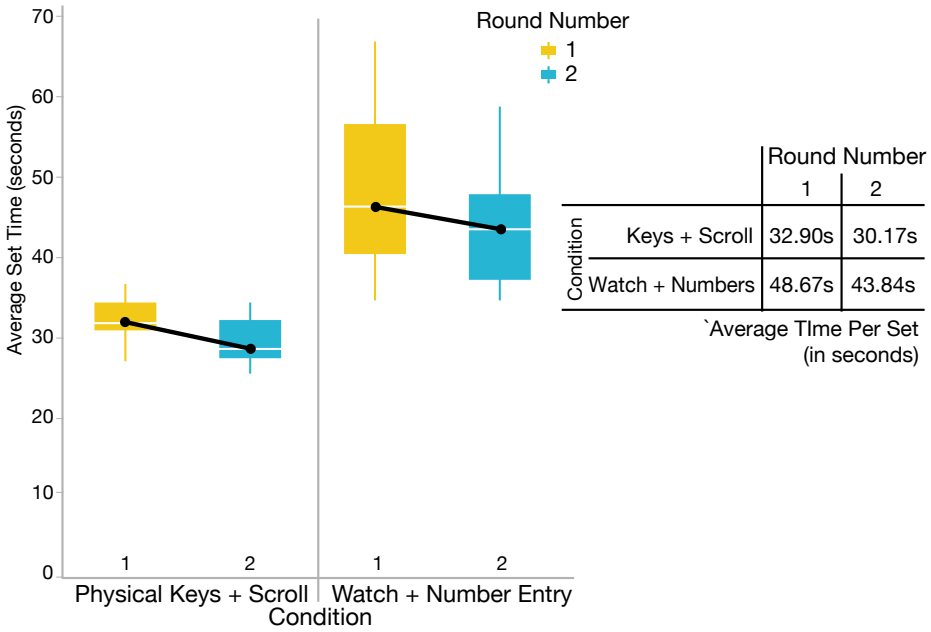


Figure 5.9: Average set times by order number, for the conditions. For both conditions, average set times were lower for order number 2 compared to that of order number 1.

Entry = 46.26 seconds, $t(18) = 8.53,\ p < 0.001$).

The effect of learning can also be observed in Figure 5.9. Participants that performed a given condition second (i.e. in round 2) were faster than participants that performed the same condition first (i.e. in round 1), irrespective of the device and task. It can also be seen that the average set times were considerably higher for number entry with swipe gestures on the smartwatch than that for scrolling with physical keys. This can be attributed to the effects of input device and task on performance. Since users were more experienced with physical keys, and in general, physical keys are known to be faster, task times were

lesser. Additionally, this could also be a result of differences in task complexity. While scrolling allowed for visual comparisons, number entry relies upon numerical comparisons.

5.7.6 Discussion

In this user study, we found a positive effect of learning BinPut with one condition, enabling skill transfer to a different task and device. Independent of the starting condition (i.e. the task–device combination used at first), users succeeded in applying BinPut to a different condition. These results support our claims that BinPut enables type-independent and device-independent input for various use scenarios. In addition, we observed that participants were able to learn the correct behaviour of the input technique in a short span of time, without any prior knowledge of the search technique. During the post-study interviews, participants commented that the change in task and device did result in some early-stage learning during both rounds. However, after experiencing BinPut in the first round, they were able to predict the input behaviour during the second round, making the usage of BinPut easier and more efficient.

Given the limited timespan of the study, we did not expect participants to become experts at the BinPut technique. For instance, the usage of the undo command is not obvious given the tree-based data structure of BinPut. More familiar linear search techniques do not require undo; they just require changing the search direction. However, we observed that 19 out of 20 participants successfully applied the undo command on several occasions. Further, some participants also

commented that after a few trials, they could learn some of the *patterns* (moves) required to reach particular values. While these aspects of learning were unexpected, they are positive indications of BinPut having a low learning threshold. These results are encouraging for BinPut as a type- and device-independent input technique.

5.8 Customising BinPut for Specific Scenarios

In its unmodified form, BinPut can be applied across a range of input types, and adopted by any device that meets the minimal requirements. To improve the usability and performance aspects for specific scenarios, BinPut supports further customisations. These customisations either reduce the number of moves required for certain input types, or improve the interaction technique for certain devices and modalities.

5.8.1 Interleaving Binary and Linear Search

We can customise BinPut to simultaneously support binary and linear search through input sets. By using binary search, users rapidly traverse the tree, while linear searching allows immediate access to neighbouring elements. A modifier key is used to switch between the search mechanisms. Alternatively, for gestural input, different gestures are used to distinguish between binary and linear search moves. For instance, a swipe gesture results in a binary search move, while scrolling gestures result in linear traversal.

However, intermittent switching between binary and linear search

moves could result in unpredictable behaviour. To tackle this, we devise a consistent scheme for the search behaviour. When users perform consecutive binary search moves, the search space is trimmed accordingly. When a linear move is triggered, the search space is re-initialised to the original (entire) input set. Consequent linear moves result in traversal through this entire search space. A subsequent binary search move now results in trimming of the original search space.

5.8.2 Unistroke Gestures

Typically, BinPut moves are discrete in nature, and repeated increment/decrement moves are performed to provide input. For devices supporting gestural input, we can also employ unistroke gestures, where one continuous stroke is used for an input task. To illustrate this, we use text entry on a touchscreen as an example.

In the original form, left and right swipes are for decrement and increment moves respectively. Thus, to perform multiple consecutive decrements or increments, multiple swipes in the same direction are required. Instead, in the modified form, we can use a series of continuous strokes for input. Vertical strokes (up and down) serve as *repetition* gestures, and replicate the previous increment or decrement stroke. The confirm command is triggered by ending the unistroke gesture, to select the input.

As illustrated in Figure 5.10, we can perform three consecutive decrements by combining a left-stroke and two vertical strokes (up + down). At this stage, a right-stroke results in an increment move.

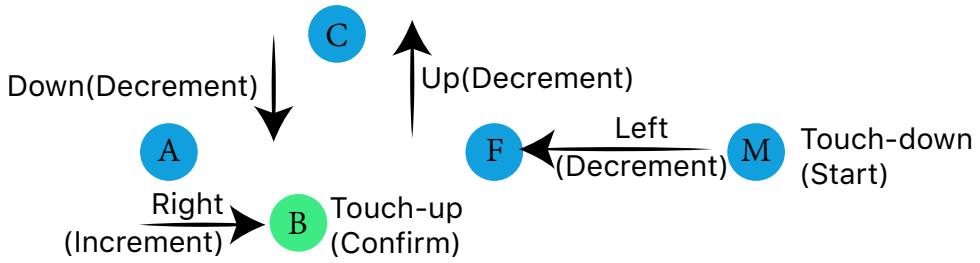


Figure 5.10: Unistroke gestures with BinPut. A continuous gesture is used to traverse the alphabet tree (Figure 5.3) and enter the character 'B'.

5.8.3 Weighted Input Sets

BinPut supports assignment of weights (or frequencies) to elements in the input set. By doing so, we can potentially reduce the number of moves required to input frequent values.

A naive approach to weighted input would be to rearrange the search tree such that important elements are placed at a higher levels. However, doing so either tampers with the inherent ordering of elements, or can lead to drastically increasing the number of levels (maximum moves), thus compromising the usability of the technique.

Instead, our weighting technique maintain the inherent ordering of elements, irrespective of usage frequencies. We insert consecutive repetitions of higher-frequency elements into the search space, hence increasing the probability of reaching these elements faster. As soon as a weighted element is encountered in the tree, all repetitions of the element are eliminated to avoid repeated encounters. It should be noted that this weighting technique offers no absolute guarantees for the number of moves required to reach particular elements.

5.9 Discussion

5.9.1 Summary

In this chapter, I presented BinPut as an input technique to facilitate the placement of input controls on post-WIMP interfaces. BinPut emphasised type- and device-independence, making it a suitable candidate for a large number of input scenarios and interfaces. In addition to the standard BinPut mechanisms, we also discussed customisations, for specific conditions, that leveraged the versatility of the input technique to further improve usability and performance. Our technical evaluation highlighted scalability aspects of BinPut, outlining the number of moves required for varying input sets, and estimating task completion time for different devices. Our user study with participants validated the transfer of learning, for the input technique, to two input tasks and with two devices. Results from our studies are encouraging for the adoption of BinPut as a ubiquitous input technique, for different post-WIMP interfaces. By providing a consistent solution across the spectrum of available and emerging devices, modalities, and input tasks, we can provide users with a consistent and scalable input technique, which is both usable and performant. Since BinPut can be applied to low-resolution devices, and alternate modalities, it can also be beneficial for accessible input needs.

5.9.2 Revisiting the Research Question

This chapter focused on facilitating the placement of input controls on post-WIMP interfaces. The research question, as introduced in the

beginning of the chapter, was:

How can we facilitate the placement of standard input controls on post-WIMP interfaces while maximising consistency across interfaces and reducing re-learning of the input technique?

Technical disparities, and limited input capabilities, offer challenges for placing typical input controls into post-WIMP interfaces. Typically, for each device or modality, a novel input technique is designed. This device-dependence reduces the scalability of input techniques to other interfaces. Additionally, users are required to re-learn different techniques when they encounter different interfaces. We addressed these challenges by presenting BinPut as a minimal input technique that could be applied by a large range of interfaces. BinPut facilitated the placement of input controls on a large range of post-WIMP interfaces by minimising the device input and output requirements, while ensuring consistent performance. The main technical contributions of this chapter are:

1. *The BinPut algorithm* for finding and entering a target element within an ordered multi-dimensional input set.
2. *BinPut type-implementations* for various types of input, including text or number entry, list selection, scrolling, and pointing.
3. *BinPut device-implementations* for different post-WIMP interfaces, such as gestural and gaze input, touchscreens, and Arduino-based sensors.
4. *Customised instances* of BinPut for specific use-cases and scenarios, to improve the performance or usability.

Our evaluations supported the technical strengths of BinPut, and pro-

vided evidence that the technique enabled transfer of learning across interfaces.

5.9.3 Principles for Placing Input Controls

The key design principles derived from this work, towards the goal of facilitating placement of input controls are:

1. **Reduce the required footprint** for input controls by minimising the number of input events.
2. **Avoid excess complexity** in the technique's implementation by reformulating input as search, and adapting simple search algorithms.
3. **Enable consistency and learning transfer** across different interfaces by reusing the same input controls and technique for different interface capabilities.

5.9.4 Limitations and Future Works

While BinPut facilitates placement of input across a diverse range of interfaces, it also has certain limitations that are important to address. Firstly, the technique is limited to ordinal input sets, and can not applied when for unordered input. Also, for small input sets ($N < 10$), BinPut does not provide significant benefits over linear techniques. Second, while BinPut aims to provide an efficient search mechanism across various scenarios, it is not fine-tuned for a particular input type or device. Other input techniques, specifically designed for certain input tasks and devices can exhibit significantly better performance than

BinPut. While BinPut can be customised to support some improvements, these customisations need to be further investigated. Third, we need to carefully consider the cognitive load of using BinPut for input tasks. While the theoretical evaluation discussed this aspect, further studies need to investigate this issue, and study the effect of learning on reducing cognition time. Taking inspiration from BinPut, future works can extend the concept of a *universal input technique*, and investigate strategies to improve performance while supporting device- and type- independence. This can further facilitate placement of input controls on new and diverse interfaces. For a new device or interface, it could provide practitioners or makers with an immediate solution to placing and enabling general-purpose input controls.

PART III

Closing

Chapter 6

Discussion

6.1 Summary of Contributions

In this dissertation, I have presented my research work on placement of interactive elements on user interfaces. As the title of this thesis suggests, the goals are twofold:
1. *Improving* how interactive elements are placed on user interfaces, more specifically GUIs.
2. *Facilitating* the placements of interactive elements on post-WIMP interfaces.

In the first part of the thesis, I addressed placement challenges in traditional **graphical interfaces**. The goal here was to improve the usability of such interfaces. I made a distinction between design-time improvements and use-time improvements, and presented two systems.

First, *Sketchplore* enabled interface designers to sketch and explore optimised placements of interface elements while designing GUI lay-

outs. It relaxed specification requirements from the designer, and automatically abstracted the design task from sketched layouts. The system used a mixed-initiative approach to suggest improvements, and did not override the designer's decisions. These factors made the system more suitable to human designers, and the fluid and uncertain nature of early-stage sketching.

Second, *Familiarisation* took a user-sided approach, and automatically improved interface layouts at use-time. It recorded a user's history, and used this to model per-user familiarity. By generating templates based on this history, it could automatically restructure new and unvisited layouts to improve visual recall time. Our studies with the two systems showed that by applying computational models to generate or restructure an interface, user performance and aesthetics of graphical layouts could be improved.

In the second part of this thesis, I addressed placement of interactions in **post-WIMP interfaces**. Novel technologies and interaction techniques supported by post-WIMP interfaces introduce additional technical complexities and challenges while creating and specifying the interface. The goal in the second part of this thesis, therefore, was to facilitate placement of interactive elements in post-WIMP interfaces.

First, I investigated the placement of interactive electronics on physical mediums. *PaperPulse* enabled non-expert users to specify and create fully-interactive paper interfaces by placing a set of electronic widgets onto visual designs. The end-to-end workflow eliminated the need for programming skills or knowledge of electronics. It facilitated the construction of customised and personalised special-purpose user interfaces. I also briefly discussed extending the concept beyond pa-

per, to other mediums such as clothing and smart homes.

Lastly, to facilitate placement of general-purpose input controls, I discussed the potential for of a universal input technique. *BinPut* presented a consistent cross-device input technique, which could be applied to different ordered input types. Using the technique, based on an adaptation of binary search, input controls could be placed on a diverse set of post-WIMP interfaces, where it might otherwise be tedious or implausible.

6.2 Research Goals and Resulting Principles

The four research questions, outlined in the introduction, and the relevant contributions of this thesis towards addressing them are summarised as follows:

1. How can we computationally support designers in the process of design exploration during early stages of placement of interactive elements on a graphical interface?

Sketchplore (chapter 2) enabled designers to simultaneously sketch and explore placement of elements on an interface with the aid of a layout optimiser. As designers placed elements on the canvas, the system suggested improvements and new layouts.

2. How can we adapt graphical interfaces for individual users' by automatically placing elements at familiar locations, at use-time, such that they are consistent with a user's mental model and enable faster visual recall?

Familiariser (chapter 3) captured user history, and modelled familiarity, to restructure new interfaces by placing elements at familiar positions. As a result, visual search could be reduced on new layouts.

3. How can we facilitate non-experts in placing interactive electronic elements on special-purpose physical interfaces by obviating the needs for programming and electronics skills?

PaperPulse (chapter 4) enabled non-expert users to place electronics onto visual designs, and specify logic. The system generated required circuits and code, and assisted the users in realising fully interactive artefacts. The chapter also briefly discussed other mediums such as wearables and home interfaces.

4. How can we facilitate the placement of standard input controls on post-WIMP interfaces while maximising consistency across interfaces and reducing re-learning of the input technique?

BinPut (chapter 5) adapted binary search to present a consistent and scalable input technique for different input types. It could be used universally to place input controls across user interfaces with varying capabilities.

Table 6.1 summarises the main contributions, corresponding research goals, and key principles.

6.3 Limitations and Future Works

This thesis was motivated by the overall goal of better supporting the processes through which user interfaces are constructed and realised. During the course of my research, I addressed some key challenges, and discussed how we could achieve this goal by improving or facilitating the placement of various interactive elements on the interface. The resulting artefacts and concepts offered promising solutions to tackling some of the research challenges. They also uncovered some

Interface Type	Contribution	Research Goal	Key Principle
Graphical User Interfaces	*Sketchplore*	Improving visual layouts at design-time	Combining sketching and exploration using a layout optimiser and predictive models
	Familiarisation	Improving visual layouts at use-time	Recording user history and modelling familiarity to automatically restructure new and unvisited designs
Post-WIMP User Interfaces	*PaperPulse*	Facilitating special-purpose interfaces with electronics	End-to-end workflow eliminating need for programming knowledge and electronics skills
	BinPut	Facilitating standard input controls across interfaces	Universal input technique with minimal requirements to support device- and type-independence

Table 6.1: Summary of contributions highlighting the research challenges, and key principles applied for each case.

limitations, and opened up promising opportunities for improvement and future research endeavours.

The first part of the thesis discussed the improvement of GUI layouts, and presented model-based tools and techniques as methods to systematically improve the interface. In our work with GUI layouts, we only addressed visual factors such as positioning, sizing, and composition. Other aspects of interaction, such as semantics between interface elements, were not considered. There is much room for future works to consider these as well, and apply similar techniques to improve not just the visual composition, but the overall interaction flow supported by GUI interfaces. Additionally, our work only addressed single-screen interfaces such as the landing page of a website. Placement issues across multiple screens was not addressed. This can partially be attributed to the characteristics and limitations of the models used for generating interface layouts. It can be beneficial to further improve computational models, to address a larger range of GUI interfaces.

In the second part of the thesis, I discussed post-WIMP and physical user interfaces. chapter 4 focussed on paper interfaces, and briefly discussed clothing and home interfaces. I presented promising end-to-end approaches to facilitate the construction of such interfaces. However, the discussion was limited to only these categories of post-WIMP interfaces. More effort needs to be invested in conceiving generalisable approaches, and a unifying framework, such that they can be applied to a more diverse set of post-WIMP interfaces. Additionally, while tools such as PaperPulse enabled users to create fully-functional interfaces, it did not support them in improving their designs. The

first part of this thesis highlighted the benefits of model-based approaches to improving graphical interfaces. It would be beneficial to apply such approaches to post-WIMP interfaces. This could aid in compensating for the lack of design knowledge of end-users, or for speeding up the design and assembly process. To achieve this, we need to further study computational models that can automatically generate and evaluate such interfaces. Personally, in the long term, I believe that hyper-customisation and personalisation will drive future improvements, making a user interface better for each person using it, and thus making UIs better for all users. Like their users, interfaces should evolve and adapt over time, constantly improving themselves. To achieve this goal, large-scale efforts will be needed to further explore how interfaces are designed, evaluated, and adapted to fit the immediate needs and requirements.

Appendix A

Nederlandstalige Samenvatting

The following is an unofficial translation of the thesis abstract. It might contain grammatical or textual errors. Please refer to the original abstract (in English) for an accurate version.

Een gebruikersinterface is het primaire gemiddelde waarmee een gebruiker interactie heeft met een computer. Interactieve elementen, geplaatst op een interface, bepalen de reikwijdte van interacties die aan gebruikers worden geboden. Dit proefschrift onderzoekt plaatsingskwesties die centraal staan in het ontwerp van gebruikersinterfaces. Het primaire doel is het ondersteunen van de constructie van gebruikersinterfaces door de plaatsing van interactieve elementen op (1) grafische gebruikersinterfaces (GUI's) en (2) post-WIMP gebruikersinterfaces te verbeteren of te vergemakkelijken.

GUI's zijn de meest gebruikte methode voor interactie met computers. Ze bestaan uit interactieve elementen die zijn georganiseerd in een visuele interface-indeling. Verbetering van de constructie van interface-lay-outs heeft een positieve invloed op de gebruiker-

sprestaties en perceptie van de interface. Het objectief verbeteren van de plaatsing van elementen is echter niet-triviaal. Het eerste deel van mijn proefschrift gaat over uitdagingen in de richting van *improvement* plaatsing op GUI-lay-outs. Hiertoe lever ik twee belangrijke bijdragen. In *Sketchplore* onderzoek ik ontwerp-tijdverbeteringen door interface-ontwerpers in staat te stellen om lay-outs te schetsen en te verkennen met behulp van een interactieve optimizer. In *Familiariser* bespreek ik een gebruikerstijdaanpak om plaatsing voor individuele gebruikers te verbeteren door beginselen van bekendheid toe te passen.

Post-WIMP-interfaces gaan verder dan het GUI-paradigma en openen nieuwe interactiemogelijkheden. Ze ondersteunen een groter aantal interactieve elementen, zoals sensoren en actuatoren. Vanwege de toegevoegde technische complexiteit, kan het een uitdaging zijn om interactieve elementen op dergelijke interfaces te plaatsen. Het tweede deel van dit proefschrift richt zich op het vergemakkelijken van de plaatsing van interactiviteit op post-WIMP-interfaces en levert twee bijdragen aan het aanpakken van plaatsingsuitdagingen. Ik onderzoek de plaatsing van interactieve elektronische elementen op fysieke interfaces. Ik presenteer *PaperPulse* als hulpmiddel voor niet-experts om elektronica op papierinterfaces te plaatsen en de discussie uit te breiden naar andere fysieke interfaces zoals wearables en smart home interfaces. In *BinPut* bespreek ik de plaatsing van standaard invoerbesturingselementen op een diverse reeks interfaces en een universele techniek die kan worden toegepast op verschillende soorten invoer en apparaten.

De concepten en principes die worden besproken in het proefschrift dragen bij aan het aanpakken van plaatsingsproblemen die cen-

traal staan in de constructie van gebruikersinterfaces. Ze kunnen resulteren in ontwerpinterfaces die performant zijn en die een breed scala aan interacties ondersteunen. Kwantitatieve en kwalitatieve evaluaties van de resulterende hulpmiddelen en technieken leveren bewijs voor de benaderingen die in dit proefschrift worden gepresenteerd.

Bibliography

[1] JONATHAN ACEITUNO, GÉRY CASIEZ, and NICOLAS ROUSSEL. How low can you go?: Human limits in small unidirectional mouse movements. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1383–1386. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2466182.

[2] JELCO ADAMCZYK, **KASHYAP TODI (ADVISOR)**, and KRIS LUYTEN (PROMOTER). Easyhome: (re)-designing your home interfaces. Bachelor Thesis. 2016.

[3] CHRISTOPH AMMA, MARCUS GEORGI, and TANJA SCHULTZ. Airwriting: Hands-free mobile text input by spotting and continuous recognition of 3d-space handwriting with inertial sensors. In *Proceedings of the 2012 16th Annual International Symposium on Wearable Computers (ISWC)*, ISWC '12, pages 52–59. IEEE Computer Society, Washington, DC, USA, 2012. ISBN 978-0-7695-4697-1. doi: 10.1109/ISWC.2012.21.

[4] JOHN R ANDERSON, D BOTHELL, C LEBIERE, and M MATESSA. An integrated theory of list memory. *Journal Of Memory And Language*, 38(4):341–380, 1998. ISSN 0749596X. doi: 10.1006/jmla.1997.2553.

[5] JOHN R ANDERSON, JON M FINCHAM, and SCOTT DOUGLASS. Practice and retention: A unifying analysis. *Journal of Experimen-*

tal Psychology-Learning Memory and Cognition, 25(5):1120–1136, 1999.

[6] GEORG APITZ and FRANÇOIS GUIMBRETIÈRE. Crossy: A crossing-based drawing application. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 3–12. ACM, New York, NY, USA, 2004. ISBN 1-58113-957-8. doi: 10.1145/1029632.1029635.

[7] YIGAL ARENS, LAWRENCE MILLER, STUART C. SHAPIRO, and NORMAN K. SONDHEIMER. Automatic construction of user-interface displays. In *Proceedings of the Seventh AAAI National Conference on Artificial Intelligence*, AAAI'88, pages 808–813. AAAI Press, 1988.

[8] SEOK-HYUNG BAE, RAVIN BALAKRISHNAN, and KARAN SINGH. Ilovesketch: As-natural-as-possible sketching system for creating 3d curve models. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 151–160. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-975-3. doi: 10.1145/1449715.1449740.

[9] GILLES BAILLY, ANTTI OULASVIRTA, TIMO KÖTZING, and SABRINA HOPPE. Menuoptimizer: Interactive optimization of menu systems. In *Proceedings of the 26th Annual ACM Symposium on User Interface Software and Technology*, UIST '13, pages 331–342. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-2268-3. doi: 10.1145/2501988.2502024.

[10] HELEN Y. BALINSKY. Evaluating interface aesthetics: measure of symmetry. In *Electronic Imaging 2006*, pages 607–608. International Society for Optics and Photonics, 2006.

[11] HELEN Y. BALINSKY, ANTHONY J. WILEY, and MATTHEW C. ROBERTS. Aesthetic measure of alignment and regularity. In *Proceedings of the 9th ACM Symposium on Document Engineering*,

DocEng '09, pages 56–65. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-575-8. doi: 10.1145/1600193.1600207.

[12] RAFAEL BALLAGAS, MEREDITH RINGEL, MAUREEN STONE, and JAN BORCHERS. istuff: A physical user interface toolkit for ubiquitous computing environments. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '03, pages 537–544. ACM, New York, NY, USA, 2003. ISBN 1-58113-630-7. doi: 10.1145/642611.642705.

[13] AYAH BDEIR and PAUL ROTHMAN. Electronics as material: Littlebits. In *Proceedings of the Sixth International Conference on Tangible, Embedded and Embodied Interaction*, TEI '12, pages 371–374. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1174-8. doi: 10.1145/2148131.2148220.

[14] ANATOLE BECK. On the linear search problem. *Israel Journal of Mathematics*, 2(4):221–228, 1964. ISSN 1565-8511. doi: 10.1007/BF02759737.

[15] BRENT BERGHMANS, AXEL FAES, MATTHIJS KAMINSKI, and **KASHYAP TODI**. Household survival: Immersive room-sized gaming using everyday objects as weapons. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 168–171. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890372.

[16] C. M. BESHERS and S. FEINER. Scope: Automated generation of graphical interfaces. In *Proceedings of the 2Nd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology*, UIST '89, pages 76–85. ACM, New York, NY, USA, 1989. ISBN 0-89791-335-3. doi: 10.1145/73660.73670.

[17] DAVID M. BLEI. Probabilistic topic models. *Commun. ACM*,

55(4):77–84, 2012. ISSN 0001-0782. doi: 10.1145/2133806.2133826.

[18] FLORIAN BLOCK, MICHAEL HALLER, HANS GELLERSEN, CARL GUTWIN, and MARK BILLINGHURST. Voodoosketch: Extending interactive surfaces with adaptable interface palettes. In *Proceedings of the 2Nd International Conference on Tangible and Embedded Interaction*, TEI '08, pages 55–58. ACM, New York, NY, USA, 2008. ISBN 978-1-60558-004-3. doi: 10.1145/1347390.1347404.

[19] FRANÇOIS BODART, ANNE-MARIE HENNEBERT, JEAN-MARIE LEHEUREUX, and JEAN VANDERDONCKT. Towards a dynamic strategy for computer-aided visual placement. In *Proceedings of the Workshop on Advanced Visual Interfaces*, AVI '94, pages 78–87. ACM, New York, NY, USA, 1994. ISBN 0-89791-733-2. doi: 10.1145/192309.192328.

[20] H. BOTTENBRUCH. Structure and use of algol 60. *J. ACM*, 9(2):161–221, 1962. ISSN 0004-5411. doi: 10.1145/321119.321120.

[21] RAINER E. BURKHARD and J. OFFERMAN. Entwurf von schreibmaschinentastaturen mittels quadratischer zuordnungsprobleme. *Operations Res*, 21:B121–B132, 1977.

[22] ALEX BUTLER, SHAHRAM IZADI, and STEVE HODGES. Sidesight: Multi-"touch" interaction around small devices. In *Proceedings of the 21st Annual ACM Symposium on User Interface Software and Technology*, UIST '08, pages 201–204. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-975-3. doi: 10.1145/1449715.1449746.

[23] BILL BUXTON. *Sketching User Experiences: Getting the Design Right and the Right Design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2007. ISBN 0123740371, 9780123740373.

[24] WILLIAM BUXTON and RICHARD SNIDERMAN. Iteration in the design of the human-computer interface. In *Proceedings of the*

13th Annual Meeting of the Human Factors Association of Canada, volume 7281, page 37. 1980.

[25] ZOYA BYLINSKII, NAM WOOK KIM, PETER O'DONOVAN, SAMI ALSHEIKH, SPANDAN MADAN, HANSPETER PFISTER, FREDO DURAND, BRYAN RUSSELL, and AARON HERTZMANN. Learning visual importance for graphic designs and data visualizations. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, UIST '17, pages 57–69. ACM, New York, NY, USA, 2017. ISBN 978-1-4503-4981-9. doi: 10.1145/3126594.3126653.

[26] D. A. CARTER and J. DIAZ. *The Elements of Pop-up: A Pop-Up Book For Aspiring Paper Engineers*. Little Simon, 1999.

[27] XIANG 'ANTHONY' CHEN, TOVI GROSSMAN, and GEORGE FITZMAURICE. Swipeboard: A text entry technique for ultra-small interfaces that supports novice to expert transitions. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 615–620. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-3069-5. doi: 10.1145/2642918.2647354.

[28] MARVIN M CHUN and YUHONG JIANG. Contextual cueing: Implicit learning and memory of visual context guides spatial attention. *Cognitive psychology*, 36(1):28–71, 1998.

[29] A. COCKBURN, P. QUINN, C. GUTWIN, G. RAMOS, and J. LOOSER. Air pointing: Design and evaluation of spatial target acquisition with and without visual feedback. *Int. J. Hum.-Comput. Stud.*, 69(6):401–414, 2011. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2011.02.005.

[30] MARCELO COELHO, LYNDL HALL, JOANNA BERZOWSKA, and PATTIE MAES. Pulp-based computing: A framework for building computers out of paper. In *CHI '09 Extended Abstracts on*

Human Factors in Computing Systems, CHI EA '09, pages 3527–3528. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-247-4. doi: 10.1145/1520340.1520525.

[31] DANIEL COHEN-OR, OLGA SORKINE, RAN GAL, TOMMER LEYVAND, and YING-QING XU. Color harmonization. In *ACM SIGGRAPH 2006 Papers*, SIGGRAPH '06, pages 624–630. ACM, New York, NY, USA, 2006. ISBN 1-59593-364-6. doi: 10.1145/1179352.1141933.

[32] NIGEL CROSS. Expertise in design: an overview. *Design Studies*, 25(5):427 – 441, 2004. ISSN 0142-694X. doi: https://doi.org/10.1016/j.destud.2004.06.002. Expertise in Design.

[33] ANIND K. DEY, RAFFAY HAMID, CHRIS BECKMANN, IAN LI, and DANIEL HSU. A cappella: Programming by demonstration of context-aware applications. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 33–40. ACM, New York, NY, USA, 2004. ISBN 1-58113-702-8. doi: 10.1145/985692.985697.

[34] MORGAN DIXON and JAMES FOGARTY. Prefab: Implementing advanced behaviors using pixel-based reverse engineering of interface structure. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '10, pages 1525–1534. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753554.

[35] AMINE DRIRA, HENRI PIERREVAL, and SONIA HAJRI-GABOUJ. Facility layout problems: A survey. *Annual Reviews in Control*, 31(2):255–267, 2007.

[36] KARIM EL BATRAN and MARK D. DUNLOP. Enhancing klm (keystroke-level model) to fit touch screen mobile devices. In *Proceedings of the 16th International Conference on Human-computer Interaction with Mobile Devices & Services*, MobileHCI '14,

pages 283–286. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-3004-6. doi: 10.1145/2628363.2628385.

[37] ELECTRONINKS INC. Paperduino 2.0 with circuit scribe. `http://www.instructables.com/id/Paperduino-20-with-Circuit-Scribe`, 2013.

[38] DANIEL FALLMAN. The new good: Exploring the potential of philosophy of technology to contribute to human-computer interaction. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 1051–1060. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979099.

[39] STEVEN K. FEINER. A grid-based approach to automating display layout. In *Proceedings on Graphics Interface '88*, pages 192–197. Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 1988.

[40] PAUL M FITTS. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of experimental psychology*, 47(6):381, 1954.

[41] ADAM FOURNEY and MICHAEL TERRY. Picl: Portable in-circuit learner. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 569–578. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1580-7. doi: 10.1145/2380116.2380188.

[42] PETER A FRENSCH. Composition during serial learning: A serial position effect. *JOURNAL OF EXPERIMENTAL PSYCHOLOGY LEARNING MEMORY AND COGNITION*, 20:423–423, 1994.

[43] KRZYSZTOF GAJOS and DANIEL S. WELD. Supple: Automatically generating user interfaces. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 93–

100. ACM, New York, NY, USA, 2004. ISBN 1-58113-815-6. doi: 10.1145/964442.964461.

[44] KRZYSZTOF Z. GAJOS, MARY CZERWINSKI, DESNEY S. TAN, and DANIEL S. WELD. Exploring the design space for adaptive graphical user interfaces. AVI '06, pages 201–208. ACM, New York, NY, USA, 2006. ISBN 1-59593-353-0. doi: 10.1145/1133265.1133306.

[45] KRZYSZTOF Z. GAJOS, JING JING LONG, and DANIEL S. WELD. Automatically generating custom user interfaces for users with physical disabilities. In *Proceedings of the 8th International ACM SIGACCESS Conference on Computers and Accessibility*, Assets '06, pages 243–244. ACM, New York, NY, USA, 2006. ISBN 1-59593-290-9. doi: 10.1145/1168987.1169036.

[46] KRZYSZTOF Z. GAJOS, JACOB O. WOBBROCK, and DANIEL S. WELD. Automatically generating user interfaces adapted to users' motor and vision capabilities. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 231–240. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-679-0. doi: 10.1145/1294211.1294253.

[47] IVÁN E. GONZÁLEZ, JACOB O. WOBBROCK, DUEN HORNG CHAU, ANDREW FAULRING, and BRAD A. MYERS. Eyes on the road, hands on the wheel: Thumb-based interaction techniques for input on steering wheels. In *Proceedings of Graphics Interface 2007*, GI '07, pages 95–102. ACM, New York, NY, USA, 2007. ISBN 978-1-56881-337-0. doi: 10.1145/1268517.1268535.

[48] SAUL GREENBERG and CHESTER FITCHETT. Phidgets: Easy development of physical interfaces through physical widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology*, UIST '01, pages 209–218. ACM, New York, NY, USA, 2001. ISBN 1-58113-438-X. doi: 10.1145/502348.502388.

[49] SAUL GREENBERG and IAN H. WITTEN. Adaptive personalized interfaces—a question of viability. *Behaviour & Information Technology*, 4(1):31–45, 1985. doi: 10.1080/01449298508901785.

[50] MARK D. GROSS and ELLEN YI-LUEN DO. Ambiguous intentions: A paper-like interface for creative design. In *Proceedings of the 9th Annual ACM Symposium on User Interface Software and Technology*, UIST '96, pages 183–192. ACM, New York, NY, USA, 1996. ISBN 0-89791-798-7. doi: 10.1145/237091.237119.

[51] TOVI GROSSMAN, XIANG ANTHONY CHEN, and GEORGE FITZMAURICE. Typing on glasses: Adapting text entry to smart eyewear. In *Proceedings of the 17th International Conference on Human-Computer Interaction with Mobile Devices and Services*, MobileHCI '15, pages 144–152. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3652-9. doi: 10.1145/2785830.2785867.

[52] TOVI GROSSMAN and GEORGE FITZMAURICE. Toolclips: An investigation of contextual video assistance for functionality understanding. In *CHI '10*, CHI '10, pages 1515–1524. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-929-9. doi: 10.1145/1753326.1753552.

[53] YOSHIKO HABUCHI, MUNEO KITAJIMA, and HARUHIKO TAKEUCHI. Comparison of eye movements in searching for easy-to-find and hard-to-find information in a hierarchically organized information structure. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08, pages 131–134. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-982-1. doi: 10.1145/1344471.1344505.

[54] PIERRE HANSEN and NENAD MLADENOVIĆ. Variable neighborhood search: Principles and applications. *European journal of operational research*, 130(3):449–467, 2001.

[55] PETER E HART, NILS J NILSSON, and BERTRAM RAPHAEL. A formal basis for the heuristic determination of minimum cost paths. *IEEE transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[56] BJÖRN HARTMANN, LEITH ABDULLA, MANAS MITTAL, and SCOTT R. KLEMMER. Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '07, pages 145–154. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-593-9. doi: 10.1145/1240624.1240646.

[57] BJÖRN HARTMANN, SCOTT R. KLEMMER, MICHAEL BERNSTEIN, LEITH ABDULLA, BRANDON BURR, AVI ROBINSON-MOSHER, and JENNIFER GEE. Reflective physical prototyping through integrated design, test, and analysis. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 299–308. ACM, New York, NY, USA, 2006. ISBN 1-59593-313-1. doi: 10.1145/1166253.1166300.

[58] MARC HASSENZAHL. The interplay of beauty, goodness, and usability in interactive products. *Hum.-Comput. Interact.*, 19(4):319–349, 2008. ISSN 0737-0024. doi: 10.1207/s15327051hci1904_2.

[59] RICHARD NA HENSON. Unchained memory: Error patterns rule out chaining models of immediate serial recall. *The Quarterly Journal of Experimental Psychology: Section A*, 49(1):80–115, 1996.

[60] STEVE HODGES, JAMES SCOTT, SUE SENTANCE, COLIN MILLER, NICOLAS VILLAR, SCARLET SCHWIDERSKI-GROSCHE, KERRY HAMMIL, and STEVEN JOHNSTON. .net gadgeteer: A new platform for k-12 computer science education. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education*,

SIGCSE '13, pages 391–396. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1868-6. doi: 10.1145/2445196.2445315.

[61] STEVE HODGES, NICOLAS VILLAR, NICHOLAS CHEN, TUSHAR CHUGH, JIE QI, DIANA NOWACKA, and YOSHIHIRO KAWAHARA. Circuit stickers: Peel-and-stick construction of interactive electronic prototypes. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1743–1746. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557150.

[62] DAVID HOLMAN and ROEL VERTEGAAL. Tactiletape: Low-cost touch sensing on curved surfaces. In *Proceedings of the 24th Annual ACM Symposium Adjunct on User Interface Software and Technology*, UIST '11 Adjunct, pages 17–18. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-1014-7. doi: 10.1145/2046396.2046406.

[63] ERIC HORVITZ. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 159–166. ACM, New York, NY, USA, 1999. ISBN 0-201-48559-1. doi: 10.1145/302979.303030.

[64] SCOTT E. HUDSON and JENNIFER MANKOFF. Rapid construction of functioning physical interfaces from cardboard, thumbtacks, tin foil and masking tape. In *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology*, UIST '06, pages 289–298. ACM, New York, NY, USA, 2006. ISBN 1-59593-313-1. doi: 10.1145/1166253.1166299.

[65] POIKA ISOKOSKI and ROOPE RAISAMO. : A rationale and an example. In *Proceedings of the Working Conference on Advanced Visual Interfaces*, AVI '00, pages 76–83. ACM, New York, NY, USA, 2000. ISBN 1-58113-252-2. doi: 10.1145/345513.345262.

[66] LAURENT ITTI and CHRISTOF KOCH. A saliency-based search mechanism for overt and covert shifts of visual attention. *Vision research*, 40(10):1489–1506, 2000.

[67] CHRISTIAN JANSSEN, ANETTE WEISBECKER, and JÜRGEN ZIEGLER. Generating user interfaces from data models and dialogue net specifications. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 418–423. ACM, New York, NY, USA, 1993. ISBN 0-89791-575-5. doi: 10.1145/169059.169335.

[68] GABE JOHNSON, MARK D. GROSS, JASON HONG, and ELLEN YI-LUEN DO. Computational support for sketching in design: A review. *Found. Trends Hum.-Comput. Interact.*, 2(1):1–93, 2009. ISSN 1551-3955. doi: 10.1561/1100000013.

[69] JUSSI P. P. JOKINEN, SAYAN SARCAR, ANTTI OULASVIRTA, CHAKLAM SILPASUWANCHAI, ZHENXIN WANG, and XIANGSHI REN. Modelling learning of new keyboard layouts. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 4203–4215. ACM, New York, NY, USA, 2017. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025580.

[70] LEVENT BURAK KARA, CHRIS M. D'ERAMO, and KENJI SHIMADA. Pen-based styling design of 3d geometry using concept sketches and template models. In *Proceedings of the 2006 ACM Symposium on Solid and Physical Modeling*, SPM '06, pages 149–160. ACM, New York, NY, USA, 2006. ISBN 1-59593-358-1. doi: 10.1145/1128888.1128909.

[71] YOSHIHIRO KAWAHARA, STEVE HODGES, BENJAMIN S. COOK, CHENG ZHANG, and GREGORY D. ABOWD. Instant inkjet circuits: Lab-based inkjet printing to support rapid prototyping of ubicomp devices. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp

'13, pages 363–372. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1770-2. doi: 10.1145/2493432.2493486.

[72] YOSHIHIRO KAWAHARA, STEVE HODGES, BENJAMIN S. COOK, CHENG ZHANG, and GREGORY D. ABOWD. Instant inkjet circuits: Lab-based inkjet printing to support rapid prototyping of ubicomp devices. In *Proceedings of the 2013 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, UbiComp '13, pages 363–372. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1770-2. doi: 10.1145/2493432.2493486.

[73] RUBAIAT HABIB KAZI, FANNY CHEVALIER, TOVI GROSSMAN, and GEORGE FITZMAURICE. Kitty: Sketching dynamic and interactive illustrations. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 395–405. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-3069-5. doi: 10.1145/2642918.2647375.

[74] KENNETH L. KELLY. Twenty-two colors of maximum contrast. *Color Engineering*, 3(26):26–27, 1965.

[75] ANDRUID KERNE, WILLIAM A. HAMILTON, and ZACHARY O. TOUPS. Culturally based design: Embodying trans-surface interaction in rummy. In *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, CSCW '12, pages 509–518. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1086-4. doi: 10.1145/2145204.2145284.

[76] DAVID E. KIERAS and ANTHONY J. HORNOF. Towards accurate and practical predictive models of active-vision-based visual search. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 3875–3884. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557324.

[77] WON CHUL KIM and JAMES D. FOLEY. Don: User interface presentation design assistant. In *Proceedings of the 3rd Annual ACM SIGGRAPH Symposium on User Interface Software and Technology*, UIST '90, pages 10–20. ACM, New York, NY, USA, 1990. ISBN 0-89791-410-4. doi: 10.1145/97924.97926.

[78] EILEEN KOWLER. Eye movements: The past 25years. *Vision research*, 51(13):1457–1483, 2011.

[79] PER-OLA KRISTENSSON and SHUMIN ZHAI. Shark2: A large vocabulary shorthand writing system for pen-based computers. In *Proceedings of the 17th Annual ACM Symposium on User Interface Software and Technology*, UIST '04, pages 43–52. ACM, New York, NY, USA, 2004. ISBN 1-58113-957-8. doi: 10.1145/1029632.1029640.

[80] HAROLD W KUHN. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 2(1-2):83–97, 1955.

[81] RANJITHA KUMAR, ARVIND SATYANARAYAN, CESAR TORRES, MAXINE LIM, SALMAN AHMAD, SCOTT R. KLEMMER, and JERRY O. TALTON. Webzeitgeist: Design mining the web. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 3083–3092. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2466420.

[82] RANJITHA KUMAR, JERRY O. TALTON, SALMAN AHMAD, and SCOTT R. KLEMMER. Bricolage: Example-based retargeting for web design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2197–2206. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979262.

[83] JAMES A. LANDAY and BRAD A. MYERS. Interactive sketching for the early stages of user interface design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI

'95, pages 43–50. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1995. ISBN 0-201-84705-1. doi: 10.1145/223904.223910.

[84] TALIA LAVIE and JOACHIM MEYER. Benefits and costs of adaptive user interfaces. *Int. J. Hum.-Comput. Stud.*, 68(8):508–524, 2010. ISSN 1071-5819. doi: 10.1016/j.ijhcs.2010.01.004.

[85] JOHNNY C. LEE, DANIEL AVRAHAMI, SCOTT E. HUDSON, JODI FORLIZZI, PAUL H. DIETZ, and DARREN LEIGH. The calder toolkit: Wired and wireless components for rapidly prototyping interactive devices. In *Proceedings of the 5th Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '04, pages 167–175. ACM, New York, NY, USA, 2004. ISBN 1-58113-787-7. doi: 10.1145/1013115.1013139.

[86] JUHA LEHIKOINEN and MIKA RÖYKKEE. Binscroll: A rapid selection technique for alphanumeric lists. In *CHI '00 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '00, pages 261–262. ACM, New York, NY, USA, 2000. ISBN 1-58113-248-4. doi: 10.1145/633292.633445.

[87] LISSA LIGHT and PETER ANDERSON. Designing better keyboards via simulated annealing. 1993.

[88] JAMES LIN, MARK W. NEWMAN, JASON I. HONG, and JAMES A. LANDAY. Denim: Finding a tighter fit between tools and practice for web site design. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '00, pages 510–517. ACM, New York, NY, USA, 2000. ISBN 1-58113-216-6. doi: 10.1145/332040.332486.

[89] SIMON LOK and STEVEN FEINER. A survey of automated layout techniques for information presentations. In *Proceedings of SmartGraphics*. 2001.

[90] SIMON LOK, STEVEN FEINER, and GARY NGAI. Evaluation of visual balance for automated layout. In *Proceedings of the 9th International Conference on Intelligent User Interfaces*, IUI '04, pages 101–108. ACM, New York, NY, USA, 2004. ISBN 1-58113-815-6. doi: 10.1145/964442.964462.

[91] SALVADOR GONZÁLEZ LÓPEZ, FRANCISCO MONTERO SIMARRO, and PASCUAL GONZÁLEZ LÓPEZ. Balores: A framework for quantitative user interface evaluation. In *New Trends in Interaction, Virtual Reality and Modeling*, pages 127–143. Springer, 2013.

[92] SUSAN LYSECKY and FRANK VAHID. Enabling nonexpert construction of basic sensor-based systems. *ACM Trans. Comput.-Hum. Interact.*, 16(1):1:1–1:28, 2009. ISSN 1073-0516. doi: 10.1145/1502800.1502801.

[93] I. SCOTT MACKENZIE, R. WILLIAM SOUKOREFF, and JOANNA HELGA. 1 thumb, 4 buttons, 20 words per minute: Design and evaluation of h4-writer. In *Proceedings of the 24th Annual ACM Symposium on User Interface Software and Technology*, UIST '11, pages 471–480. ACM, New York, NY, USA, 2011. ISBN 978-1-4503-0716-1. doi: 10.1145/2047196.2047258.

[94] SCOTT I. MACKENZIE. Fitts' law as a research and design tool in human-computer interaction. *Human–Computer Interaction*, 7(1):91–139, 1992.

[95] ATSUHIKO MAEDA, HIROHITO INAGAKI, and MASANOBU ABE. Arrow tag: A direction-key-based technique for rapidly selecting hyperlinks while gazing at a screen. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '09, pages 1025–1028. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-246-7. doi: 10.1145/1518701.1518857.

[96] ANDERS MARKUSSEN, MIKKEL RØNNE JAKOBSEN, and KASPER HORNBÆK. Vulture: A mid-air word-gesture keyboard. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 1073–1082. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2556964.

[97] BARBARA J. MEIER, ANNE MORGAN SPALTER, and DAVID B. KARELITZ. Interactive color palette tools. *IEEE Comput. Graph. Appl.*, 24(3):64–72, 2004. ISSN 0272-1716. doi: 10.1109/MCG.2004.1297012.

[98] DAVID A. MELLIS, SAM JACOBY, LEAH BUECHLEY, HANNAH PERNER-WILSON, and JIE QI. Microcontrollers as material: Crafting circuits with paper, conductive ink, electronic components, and an "untoolkit". In *Proceedings of the 7th International Conference on Tangible, Embedded and Embodied Interaction*, TEI '13, pages 83–90. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1898-3. doi: 10.1145/2460625.2460638.

[99] BRAD MYERS, SCOTT E. HUDSON, and RANDY PAUSCH. Past, present, and future of user interface software tools. *ACM Trans. Comput.-Hum. Interact.*, 7(1):3–28, 2000. ISSN 1073-0516. doi: 10.1145/344949.344959.

[100] BRAD A. MYERS, RISHI BHATNAGAR, JEFFREY NICHOLS, CHOON HONG PECK, DAVE KONG, ROBERT MILLER, and A. CHRIS LONG. Interacting at a distance: Measuring the performance of laser pointers and other devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '02, pages 33–40. ACM, New York, NY, USA, 2002. ISBN 1-58113-453-3. doi: 10.1145/503376.503383.

[101] MARK W. NEWMAN and JAMES A. LANDAY. Sitemaps, storyboards, and specifications: A sketch of web site design practice.

In *Proceedings of the 3rd Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques*, DIS '00, pages 263–274. ACM, New York, NY, USA, 2000. ISBN 1-58113-219-0. doi: 10.1145/347642.347758.

[102] JEFFREY NICHOLS and TESSA LAU. Mobilization by demonstration: Using traces to re-author existing web sites. In *Proceedings of the 13th International Conference on Intelligent User Interfaces*, IUI '08, pages 149–158. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-987-6. doi: 10.1145/1378773.1378793.

[103] JEFFREY NICHOLS, BRAD A. MYERS, MICHAEL HIGGINS, JOSEPH HUGHES, THOMAS K. HARRIS, RONI ROSENFELD, and MATHILDE PIGNOL. Generating remote control interfaces for complex appliances. In *Proceedings of the 15th Annual ACM Symposium on User Interface Software and Technology*, UIST '02, pages 161–170. ACM, New York, NY, USA, 2002. ISBN 1-58113-488-6. doi: 10.1145/571985.572008.

[104] JEFFREY NICHOLS, BRAD A. MYERS, and BRANDON ROTHROCK. Uniform: Automatically generating consistent remote control user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '06, pages 611–620. ACM, New York, NY, USA, 2006. ISBN 1-59593-372-7. doi: 10.1145/1124772.1124865.

[105] JAKOB NIELSEN. *Usability Engineering*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1993. ISBN 0125184050.

[106] DONALD A. NORMAN. *The Design of Everyday Things*. Basic Books, Inc., New York, NY, USA, 2002. ISBN 9780465067107.

[107] PETER ODONOVAN, ASEEM AGARWALA, and AARON HERTZMANN. Learning layouts for single-pagegraphic designs. *IEEE Transactions on Visualization and Computer Graphics*, 20(8):1200–1213, 2014. ISSN 1077-2626. doi: 10.1109/TVCG.2014.48.

[108] PETER O'DONOVAN, ASEEM AGARWALA, and AARON HERTZMANN. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1221–1224. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702149.

[109] D. R. OLSEN, JR. A programming language basis for user interface. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '89, pages 171–176. ACM, New York, NY, USA, 1989. ISBN 0-89791-301-9. doi: 10.1145/67449.67485.

[110] STEPHEN ONEY, CHRIS HARRISON, AMY OGAN, and JASON WIESE. Zoomboard: A diminutive qwerty soft keyboard using iterative zooming for ultra-small devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2799–2802. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2481387.

[111] ANTTI OULASVIRTA, ANNA REICHEL, WENBIN LI, YAN ZHANG, MYROSLAV BACHYNSKYI, KEITH VERTANEN, and PER OLA KRISTENSSON. Improving two-thumb text entry on touchscreen devices. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2765–2774. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2481383.

[112] STEVEN PEETERS, **KASHYAP TODI (ADVISOR)**, and KRIS LUYTEN (PROMOTER). The home logging toolkit. Bachelor Thesis. 2017.

[113] HANNAH PERNER-WILSON, LEAH BUECHLEY, and MIKA SATOMI. Handcrafting textile interfaces from a kit-of-no-parts. In *Proceedings of the Fifth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '11, pages 61–68. ACM, New

York, NY, USA, 2011. ISBN 978-1-4503-0478-8. doi: 10.1145/1935701.1935715.

[114] SIMON T. PERRAULT, ERIC LECOLINET, JAMES EAGAN, and YVES GUIARD. Watchit: Simple gestures and eyes-free interaction for wristwatches and bracelets. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 1451–1460. ACM, New York, NY, USA, 2013. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2466192.

[115] ANGEL R. PUERTA, HENRIK ERIKSSON, JOHN H. GENNARI, and MARK A. MUSEN. Model-based automated generation of user interfaces. In *Proceedings of the Twelfth National Conference on Artificial Intelligence (Vol. 1)*, AAAI '94, pages 471–477. American Association for Artificial Intelligence, Menlo Park, CA, USA, 1994. ISBN 0-262-61102-3.

[116] JIE QI and LEAH BUECHLEY. Electronic popables: Exploring paper-based computing through an interactive pop-up book. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10, pages 121–128. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-841-4. doi: 10.1145/1709886.1709909.

[117] JIE QI and LEAH BUECHLEY. Sketching in circuits: Designing and building electronics on paper. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 1713–1722. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557391.

[118] RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2457–2466. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702487.

[119] RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *SIGGRAPH 2015: Studio*, SIGGRAPH '15, pages 3:1–3:1. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3637-6. doi: 10.1145/2785585.2792694.

[120] RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach for embedding electronics in paper designs. In *ACM SIGGRAPH 2015 Posters*, SIGGRAPH '15, pages 9:1–9:1. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3632-1. doi: 10.1145/2787626.2792650.

[121] RAF RAMAKERS, **KASHYAP TODI**, and KRIS LUYTEN. Paperpulse: An integrated approach to fabricating interactive paper. In *Proceedings of the 33rd Annual ACM Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '15, pages 267–270. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3146-3. doi: 10.1145/2702613.2725430.

[122] KEITH RAYNER. The 35th sir frederick bartlett lecture: Eye movements and attention in reading, scene perception, and visual search. *Quarterly Journal of Experimental Psychology*, 62(8):1457–1506, 2009.

[123] JOHN RHEINFRANK and SHELLEY EVENSON. Design languages. In *Bringing design to software*, pages 63–85. ACM, 1996.

[124] RUTH ROSENHOLTZ, AMAL DORAI, and ROSALIND FREEMAN. Do predictions of visual perception aid design? *ACM Trans. Appl. Percept.*, 8(2):12:1–12:20, 2011. ISSN 1544-3558. doi: 10.1145/1870076.1870080.

[125] RUTH ROSENHOLTZ, YUANZHEN LI, and LISA NAKANO. Measuring visual clutter. *Journal of vision*, 7(2):17, 2007.

[126] DARIO D SALVUCCI. An integrated model of eye movements and visual encoding. *Cogn. Syst. Res.*, 1(4):201–220, 2001. ISSN 1389-0417. doi: 10.1016/S1389-0417(00)00015-2.

[127] SAYAN SARCAR, JUSSI JOKINEN, ANTTI OULASVIRTA, XIANGSHI REN, CHAKLAM SILPASUWANCHAI, and ZHENXIN WANG. Ability-based optimization: Designing smartphone text entry interface for older adults. *IEEE Pervasive Computing*, 2018.

[128] GREG SAUL, CHENG XU, and MARK D. GROSS. Interactive paper devices: End-user design & fabrication. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10, pages 205–212. ACM, New York, NY, USA, 2010. ISBN 978-1-60558-841-4. doi: 10.1145/1709886.1709924.

[129] VALKYRIE SAVAGE, XIAOHAN ZHANG, and BJÖRN HARTMANN. Midas: Fabricating custom capacitive touch sensors to prototype interactive objects. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, UIST '12, pages 579–588. ACM, New York, NY, USA, 2012. ISBN 978-1-4503-1580-7. doi: 10.1145/2380116.2380189.

[130] DOMINIK SCHMIDT, RAF RAMAKERS, ESBEN W. PEDERSEN, JOHANNES JASPER, SVEN KÖHLER, AILEEN POHL, HANNES RANTZSCH, ANDREAS RAU, PATRICK SCHMIDT, CHRISTOPH STERZ, YANINA YURCHENKO, and PATRICK BAUDISCH. Kickables: Tangibles for feet. In *Proceedings of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI '14, pages 3143–3152. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557016.

[131] MARTIN SCHMITZ, MOHAMMADREZA KHALILBEIGI, MATTHIAS BALWIERZ, ROMAN LISSERMANN, MAX MÜHLHÄUSER, and JÜRGEN STEIMLE. Capricate: A fabrication pipeline to design and 3d print capacitive touch sensors

for interactive objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15, pages 253–258. ACM, New York, NY, USA, 2015. ISBN 978-1-4503-3779-3. doi: 10.1145/2807442.2807503.

[132] A. SEARS. Layout appropriateness: a metric for evaluating user interface widget layout. *IEEE Transactions on Software Engineering*, 19(7):707–719, 1993. ISSN 0098-5589. doi: 10.1109/32.238571.

[133] ANDREW SEARS, JULIE A JACKO, JOSEY CHU, and FRANCISCO MORO. The role of visual search in the design of effective soft keyboards. *Behaviour & Information Technology*, 20(3):159–166, 2001.

[134] CLAYTON SHEPARD, AHMAD RAHMATI, CHAD TOSSELL, LIN ZHONG, and PHILLIP KORTUM. Livelab: Measuring wireless networks and smartphone users in the field. *SIGMETRICS Perform. Eval. Rev.*, 38(3):15–20, 2011. ISSN 0163-5999. doi: 10.1145/1925019.1925023.

[135] MICHAEL SHORTER, JON ROGERS, and JOHN MCGHEE. Enhancing everyday paper interactions with paper circuits. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 39–42. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2902-6. doi: 10.1145/2598510.2598584.

[136] SURYA P. SINGH and RENDUCHINTALA RK SHARMA. A review of different approaches to the facility layout problems. *The International Journal of Advanced Manufacturing Technology*, 30(5-6):425–433, 2006.

[137] IVAN E. SUTHERLAND. Sketchpad: A man-machine graphical communication system. In *Proceedings of the May 21-23, 1963,*

Spring Joint Computer Conference, AFIPS '63 (Spring), pages 329–346. ACM, New York, NY, USA, 1963. doi: 10.1145/1461551.1461591.

[138] LUKE SWARTZ. Overwhelmed by technology: How did user interface failures on board the uss vincennes lead to 290 dead. *Erişim tarihi*, 25, 2001.

[139] MICHAEL TERRY, ELIZABETH D. MYNATT, KUMIYO NAKAKOJI, and YASUHIRO YAMAMOTO. Variation in element and action: Supporting simultaneous development of alternative solutions. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '04, pages 711–718. ACM, New York, NY, USA, 2004. ISBN 1-58113-702-8. doi: 10.1145/985692.985782.

[140] **KASHYAP TODI**, DONALD DEGRAEN, BRENT BERGHMANS, AXEL FAES, MATTHIJS KAMINSKI, and KRIS LUYTEN. Purpose-centric appropriation of everyday objects as game controllers. In *Proceedings of the CHI '16 Extended Abstracts*, CHI EA '16, pages 2744–2750. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2892448.

[141] **KASHYAP TODI**, JUSSI JOKINEN, KRIS LUYTEN, and ANTTI OULASVIRTA. Familiarisation: Restructuring layouts with visual learning models. In *Proceedings of the 2018 ACM Conference on Interactive User Interfaces*, IUI '18. ACM, New York, NY, USA, 2018.

[142] **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Enabling eyes-free interactions on jacket buttons. In *Proceedings of the Extended Abstracts of the 32Nd Annual ACM Conference on Human Factors in Computing Systems*, CHI EA '14, pages 1549–1554. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2474-8. doi: 10.1145/2559206.2581155.

[143] **KASHYAP TODI** and KRIS LUYTEN. Suit up!: Inconspicuous interactions on jacket buttons. In *Proceedings of the 2014 CHI Conference Workshop on Inconspicuous Interactions*, CHI EA '14. ACM, New York, NY, USA, 2014.

[144] **KASHYAP TODI**, KRIS LUYTEN, and ANDREW VANDE MOERE. Making smart homes personal: Fabrication and customisation of home interfaces. In *Proceedings of the CHI '15 Workshop on Smart for Life: Designing Smart Home Technologies that Evolve with Users*, CHI EA '15. 2015.

[145] **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore layout designs with an optimiser. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 3780–3783. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4082-3. doi: 10.1145/2851581.2890236.

[146] **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplore: Sketch and explore with a layout optimiser. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, DIS '16, pages 543–555. ACM, New York, NY, USA, 2016. ISBN 978-1-4503-4031-1. doi: 10.1145/2901790.2901817.

[147] **KASHYAP TODI**, DARYL WEIR, and ANTTI OULASVIRTA. Sketchplorer: A mixed-initiative tool for sketching and exploring interactive layout designs. In *Proceedings of the CHI '17 Workshop on Mixed-Initiative Creative Interfaces*. 2017.

[148] OUTI TUISKU, PÄIVI MAJARANTA, POIKA ISOKOSKI, and KARI-JOUKO RÄIHÄ. Now dasher! dash away!: Longitudinal study of fast text entry by eye gaze. In *Proceedings of the 2008 Symposium on Eye Tracking Research & Applications*, ETRA '08, pages 19–26. ACM, New York, NY, USA, 2008. ISBN 978-1-59593-982-1. doi: 10.1145/1344471.1344476.

[149] ANDRIES VAN DAM. Post-wimp user interfaces. *Communications of the ACM*, 40(2):63–67, 1997.

[150] MARC VAN DROOGENBROECK and SÉBASTIEN PIÉRARD. Object descriptors based on a list of rectangles: Method and algorithm. In *Proceedings of the 10th International Conference on Mathematical Morphology and Its Applications to Image and Signal Processing*, ISMM'11, pages 155–165. Springer-Verlag, Berlin, Heidelberg, 2011. ISBN 978-3-642-21568-1.

[151] MARTIJN VAN WELIE and GERRIT C VAN DER VEER. Pattern languages in interaction design: Structure and organization. In *Proceedings of interact*, volume 3, pages 1–5. 2003.

[152] JEAN M. VANDERDONCKT and FRANÇOIS BODART. Encapsulating knowledge for intelligent automatic interaction objects selection. In *Proceedings of the INTERACT '93 and CHI '93 Conference on Human Factors in Computing Systems*, CHI '93, pages 424–429. ACM, New York, NY, USA, 1993. ISBN 0-89791-575-5. doi: 10.1145/169059.169340.

[153] VASILIS VLACHOKYRIAKOS, ROB COMBER, KARIM LADHA, NICK TAYLOR, PAUL DUNPHY, PATRICK MCCORRY, and PATRICK OLIVIER. Postervote: Expanding the action repertoire for local political activism. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 795–804. ACM, New York, NY, USA, 2014. ISBN 978-1-4503-2902-6. doi: 10.1145/2598510.2598523.

[154] DANIEL VOGEL and RAVIN BALAKRISHNAN. Distant freehand pointing and clicking on very large, high resolution displays. In *Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology*, UIST '05, pages 33–42. ACM, New York, NY, USA, 2005. ISBN 1-59593-271-2. doi: 10.1145/1095034.1095041.

[155] DAVID J. WARD, ALAN F. BLACKWELL, and DAVID J. C. MACKAY. Dasher—a data entry interface using continuous gestures and language models. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology*, UIST '00, pages 129–137. ACM, New York, NY, USA, 2000. ISBN 1-58113-212-3. doi: 10.1145/354401.354427.

[156] MARK WEISER. The computer for the 21 st century. *Scientific american*, 265(3):94–105, 1991.

[157] MAX WERTHEIMER. A brief introduction to gestalt, identifying key theories and principles. *Psychol Forsch*, 4:301–350, 1923.

[158] L.G. WILLIAMS. A study of visual search using eye movement recordings. Technical report, DTIC Document, 1966.

[159] JACOB O. WOBBROCK, SHAUN K. KANE, KRZYSZTOF Z. GAJOS, SUSUMU HARADA, and JON FROEHLICH. Ability-based design: Concept, principles and examples. *ACM Trans. Access. Comput.*, 3(3):9:1–9:27, 2011. ISSN 1936-7228. doi: 10.1145/1952383.1952384.

[160] JACOB O. WOBBROCK, BRAD A. MYERS, and JOHN A. KEMBEL. Edgewrite: A stylus-based text entry method designed for high accuracy and stability of motion. In *Proceedings of the 16th Annual ACM Symposium on User Interface Software and Technology*, UIST '03, pages 61–70. ACM, New York, NY, USA, 2003. ISBN 1-58113-636-6. doi: 10.1145/964696.964703.

[161] JACOB O. WOBBROCK, ANDREW D. WILSON, and YANG LI. Gestures without libraries, toolkits or training: A $1 recognizer for user interface prototypes. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 159–168. ACM, New York, NY, USA, 2007. ISBN 978-1-59593-679-0. doi: 10.1145/1294211.1294238.

[162] YIN YIN WONG. Rough and ready prototypes: Lessons from graphic design. In *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems*, CHI '92, pages 83–84. ACM, New York, NY, USA, 1992. doi: 10.1145/1125021.1125094.

[163] SHENGXIANG YANG and XIN YAO. Population-based incremental learning with associative memory for dynamic environments. *Evolutionary Computation, IEEE Transactions on*, 12(5):542–561, 2008.

[164] YEONSOO YANG and SCOTT R. KLEMMER. Aesthetics matter: Leveraging design heuristics to synthesize visually satisfying handheld interfaces. In *CHI '09 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '09, pages 4183–4188. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-247-4. doi: 10.1145/1520340.1520637.

[165] TOM YEH, TSUNG-HSIANG CHANG, and ROBERT C. MILLER. Sikuli: Using gui screenshots for search and automation. In *Proceedings of the 22Nd Annual ACM Symposium on User Interface Software and Technology*, UIST '09, pages 183–192. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-745-5. doi: 10.1145/1622176.1622213.

[166] MATHIEU ZEN and JEAN VANDERDONCKT. Towards an evaluation of graphical user interfaces aesthetics based on metrics. In *Research Challenges in Information Science (RCIS), 2014 IEEE Eighth International Conference on*, pages 1–12. IEEE, 2014.

[167] SHUMIN ZHAI, MICHAEL HUNTER, and BARTON A. SMITH. Performance optimization of virtual keyboards. *Human–Computer Interaction*, 17(2-3):229–269, 2002.

[168] G.K. ZIPF. *Human behavior and the principle of least effort: an introduction to human ecology*. Addison-Wesley Press, 1949.