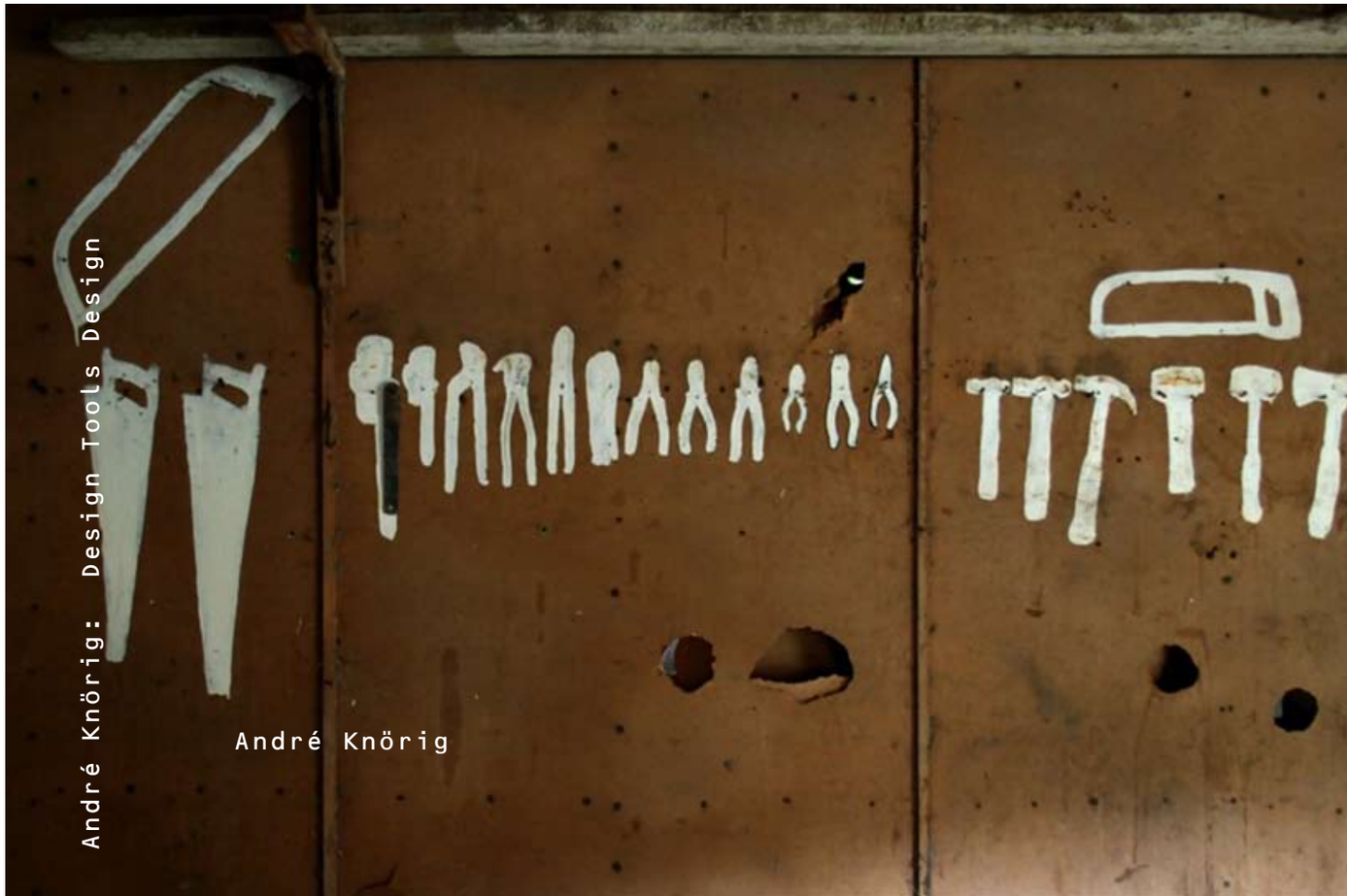
Design Tools Design

How to design tools for designers, and a proposal of two new tools for the design of physical interactions



Master Thesis, FH Potsdam, 2008

André Knörig

André Knörig: Design Tools Design

Design Tools Design

How to design tools for designers, and a proposal of two new tools for the design of physical interactions

André Knörig

Title photo by Lynn Irving
http://flickr.com/photos/spiderpops/202001463

submitted by
André Knörig

to the University of Applied Sciences Potsdam
in partial fulfillment of the requirements for the degree
Master of Design (Interface Design)

August 2008

supervised by

Boris Müller
Professor of Interface Design
University of Applied Sciences Potsdam

Reto Wettach
Professor of Interface Design
University of Applied Sciences Potsdam

ABSTRACT

This thesis investigates the use of tools, specifically software tools, in the design of user interfaces. Based on an analysis of the design process and existing tools, it digests a set of guidelines for the creation of such tools. Two relevant tools for the design of physical interactions are proposed as a consequence, one for the early stage of sketching, the other for the later stage of production.

Software tools for design have a long and successful history, with broad support for nearly every design discipline. This thesis argues that a successful "design tool design" requires a thorough understanding of the process and activities the designers are involved in. Based on this understanding and a survey of existing tools for the discipline of interaction design, a set of guidelines is established. The three main principles of creativity, craftsmanship, and practicability attempt to harmonize findings from research and commercial applications.

Finally, this thesis contributes two new tools to the young domain of physical interaction design. Both fill important gaps in adequate tool support. Sketchbook provides assistance in the early conceptual and sketching phases, bringing structure and interactivity to physical sketches. Its informal approach lets the designer gradually evolve his ideas, to the stage where they can be directly fed into electronic prototyping toolkits.

Fritzing enables designers to bring their prototypes to a higher level of fidelity. With little technical knowledge, they can turn their breadboard-based electronics into professionally produced PCBs and move one step further to self-production. Fritzing also makes documentation and sharing for the first time feasible for designers, providing a potential creativity boost to the community.

ACKNOWLEDGMENTS

I would like to express my gratitude to a number of kind people who have supported and accompanied me through this endeavor:

My supervisors, Prof. Boris Müller and Prof. Reto Wettach, for their wise critiques and motivating challenges.

My study mates, especially Christian Behrens, Johannes Landstorfer, and Larissa Pschetz, for their honest words and their shared understanding of the life as a master student.

The Fritzing team, Jonathan Cohen, Zach Eveland, Dirk van Oosterbosch, Omer Yosha, Travis Robertson, and as of late, Brendan Howell, Mariano Crowe, Stefan Hermann and Marcus Paeschke, for their enthusiastic working attitude and the highly enjoyable atmosphere. And of course the MWFK Brandenburg for sponsoring this research project.

Also all the participants of the Fritzing workshops for their passionate discussions and contributions.

Dr. Eva Hornecker and Prof. Frank Heidmann for helpful pointers to relevant research.

Fabian Hemmert, Julia Werner, Hans Kadel, and Dr. Gesche Joost for the exciting work on the research project that also led to the "Dynamic Knobs" prototype discussed here.

Finally, my family and Sabrina's family for their deep and totally uncritical support.

Sabrina, for being her and being with me.

EIDESSTATTLICHE ERKLÄRUNG

Ich erkläre hiermit an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe; die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht. Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungskommission vorgelegt und auch nicht veröffentlicht.

André Knörig
Berlin, den 1. September 2008

TABLE OF CONTENTS

CONTEXT 1

If all you have is a hammer,
everything looks like a nail.

(Folk wisdom)

The computer has become the dominant tool of our times, the jack-of-all-trades which assists us with any problem. Its power is abstraction, and the creativity of its users brings us new practical (and unpractical) applications every day. In his influential essays, the critical computer scientist Fred Brooks repeatedly reminds his colleagues that they are "toolsmiths" much rather than scientists (Brooks, 1977 & 1994).

We should therefore expect that by now, fourty years after the invention of the desktop computer, we are near-optimally supported by computer tools. In my diploma thesis (Knörig, 2006) I have already shown that existing tools are still ignoring an essential part of our creativity: The human body is largely neglected when it comes to interacting with computers. My belief is that this harms our creativity, especially since the desktop computer has a tendency to absorb our full attention (ibid., p. 36).

This thesis, however, starts by accepting the dominance of the graphical user interface and will look into the tools that are available there. The domain however, to which these tools shall be applied, is again bodily. It is about the design of physical, bodily interactions between human beings and electronic artefacts. In other words, how can software tools help to design new interactive hardware?

1.1 DESIGN TOOLS

Tools have always been a powerful propeller of human evolution. In fact, until the late eighteenth century, the use of tools was regarded as the main

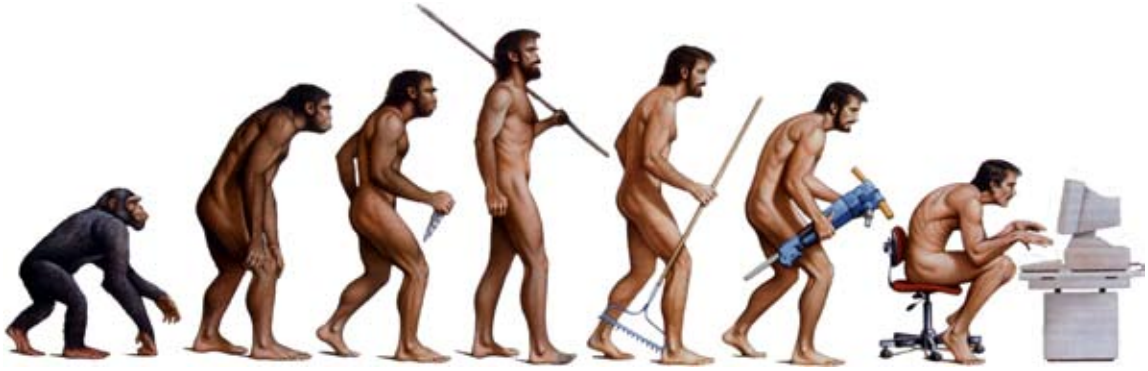


Figure 1.1: The influence of tools on the evolution of mankind
Drawing by Braldt Bralds.

distinguishing property between humans and animals. This view can no longer be held and is now more differentiated. Human tool use is in principle much more advanced, e.g., we use tools to produce other tools, we retain them for repeated use, we build tool-sets for special domains and even specialized workshops (Greenberg, 1993, ch. 1).

Tools are artefacts that are made to *extend* human abilities. Classically, tools are pragmatic extensions for the hand to let us control and manipulate the physical environment (Britannica, 2008). The German word for tool, *Werkzeug*, literally means "stuff to work with" or "stuff to create a work with". This definition has broadened with the development of human culture. For instance, we have created tools for learning and collaborating. The computer has widened this definition even further. It has stimulated the creation of an endless range of tools for any purpose, one of the first ideas being Vannevar Bush's mind amplification tool, the memex (Bush, 1945). If one accepts the synonym of "software tool" for "software application", then the computer is certainly the tool with which the most other tools, and also the most complex, were ever built.

In the domain of design, tools have always played a major role, because design is all about manipulating the physical environment. It is necessary to make a distinction between different purposes of tools here. The process of design reaches from early conceptual stages through evolving stages of sketches and prototypes to the planning for production, nowadays often including production itself. The tools used are roughly related to these stages. It is also necessary to distinguish tools from methods. The word *tool* is often mistakenly used for what is actually a *method*. A method is an

Figure 1.2: Toolset of a Make-Up Artist
Photo by the Author

instruction of how to go about doing something, and can involve tools. For instance, Brainstorming is a method that involves pens and paper as tools.

The oldest and most powerful conceptual tool is, and likely will remain, the pen(cil) (Figure 1.3). It gives complete freedom, and every child knows how to use it to express its ideas. Consequently, it is the most widely used tool in any conceptual work, and this especially holds for design, if one is

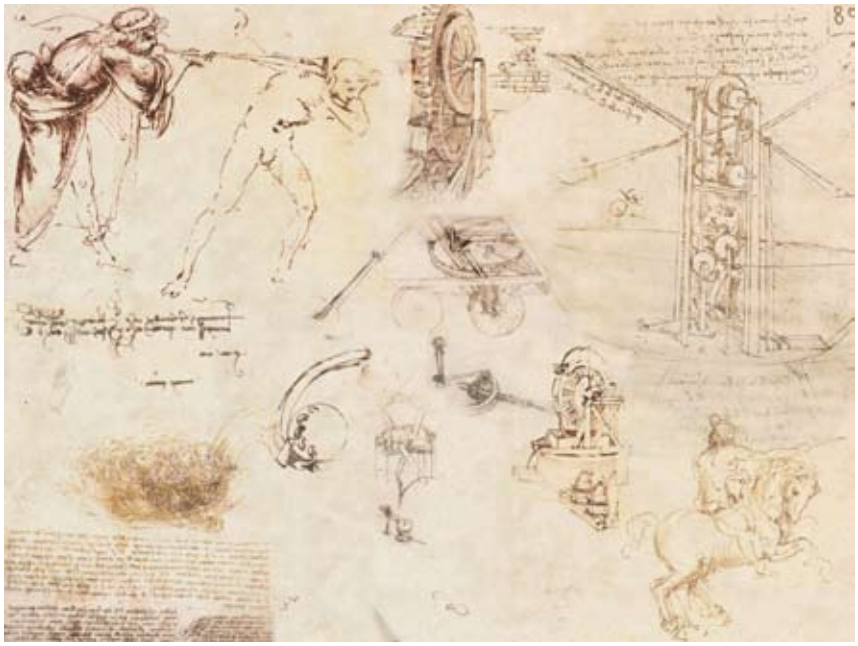


Figure 1.3: The pencil as a tool for exploring ideas
From Leonardo Da Vinci's sketchbook, around 1500.

not working with the production material directly. The pen enables us to quickly, with minimal cost and effort, try out ideas, communicate them, change and either discard or refine them. It is thus ideally suited for the early stages of design, when ideas are quick and plentiful.

The pencil is also the most general tool, used across all disciplines. The more the design process progresses, the more specialized the tools become. Design is closely related to the available production technologies and therefore the furniture designer uses different tools than the jewelry designer to experiment with designs. Every designer uses a domain-specific

(sometimes even individual) toolset that is constantly honed and extended (Figure 1.2 shows an example), and the designer's skills in applying these tools are constantly improved through use, often to an extreme mastery.

While tools remain a practical means to an end, this close connection with the tool also causes a dependency. Not only does the tool set the constraints for what we can design, it also defines how we perceive our work: It structures our approach and determines how much effort it will consume.

On a larger level, our workplace is shaped by the tools we use, and even our working conditions (Abercrombie & Glaser, 1997). When a new tool enters the stage, all of these circumstances will be transformed. And if it is a dramatic change, as we are witnessing it with the introduction of computers, it affects our whole work-life, where we live, when we work and how we work together. We should thus be critical about the tools we employ.

1.2 COMPUTER AS DESIGN TOOL

Take the example of photographic design: As with other fields of design the digitalization has transformed it from the ground up. Digitalization itself is a new technology rather than a new tool, but it came to us in the form of new tools. Not only do photographers now use digital cameras instead of analog ones: Every changed aspect of the tools affects the way photos are designed today. The large memory of the digital camera and the ability to directly view the pictures taken has led to a different style of shooting: It enables the photographer to be more flexible, experiment more, and gives him more time to take pictures. On the other hand, it leads to less concentrated and focused work because of the constant checking, and discussing with clients.

Maybe even more importantly, the postproduction is now in the hands of the designer. Retouching software like Photoshop® provides him with endless creative freedom in manipulating a picture after it was taken. This is an extreme change over the dark-room. Professional photographers now shoot their pictures with the retouching steps in mind, so that a large percentage of the photographic design now happens at the computer. Much more than a changed workflow, these tools have changed the viewing habits of a whole generation, to the extent that photography has lost its status of documenting the reality. Finally, digital tools have made photography cheaper and less time-consuming, and therefore accessible to anyone.

From today's perspective, the computer as a tool has brought mainly advantages for designers:

- More freedom and flexibility: A much larger design space, easier and cheaper experimentation
- More focus: Less extraneous and less repetitive manual work
- More productive: Eased re-use, variation and multiplication
- More control over the design: Working with a nearly perfect simulation of the final product, and ability to create proofs
- More self-responsibility: Designer as full-service agent from concept to production

As we could already see with photography, advantages come at a cost. In general, the accessibility of design tools has certainly led to a degradation of quality overall, and the freedom of expression to a certain arbitrariness. Design has become a mass phenomenon, it is less exclusive. But maybe that is an advantage, after all (see also Schneider, 2005, ch. 16).

Figure 1.4: Radical black box design for a tv set
Black 201 Television Set for Brion Vega, by Richard Sapper and Marco Zanuso, 1969

1.3 (PHYSICAL) INTERACTION DESIGN

Besides revolutionizing the classic design disciplines, computers brought with them a totally new area of design: computers. Or rather, electronically equipped products. How do these objects look like, how do they appear, when their function does not imply any form (Figure 1.4)? How do we use them, when they have no physical function? This design need led to a new discipline named interaction design. Due to its relatively short history and potentially broad scope, there is no common definition for it. I will assume Gillian Crampton-Smith's definition here (Moggridge, 2006, p. xi):

> If I were to sum up interaction design in a sentence, I would say that it's about shaping our everyday life through digital artifacts for work, for play, and for entertainment.

As Löwgren (2008) points out, the term has at least two traditions, one from the academic, computer science-related area of Human-Computer Interaction, the other from a design background. This thesis is mainly concerned with the latter, i.e., how we design better digital artifacts on a human experiential level that includes, besides functionality, aesthetic and ethical dimensions.

This rather broad definition also means that it goes beyond just Graphical User Interfaces (GUIs). Because of the de-facto standard user interface established by the desktop computer (and its variants), interaction design is often mistaken as being occupied with just what happens on the screen. On the other hand, the term tangible interaction design refers to non-GUI interfaces, but only those which are based on tangibility. For lack of a term to refer to a broader definition, I am sometimes using the term physical interaction design here.

A classic example will help to illustrate this further. One of the first and very convincing concepts for "physical" interaction design (in this case also tangible) is the marble answering machine by Durrell Bishop (1992, Figure 1.5). An incoming voice message is represented as a marble and delivered by the machine. Playing it back is a matter of putting it into the playback bowl, and a marble is freed by putting it back into the container. Important messages can be kept and treasured. The marble answering machine is an excellent example for the freedom we gain with the form-lessness of digital technology. Instead of just wrapping a grey box around it, we can design the form and interactions towards human interests.

This enormous freedom is not easily turned into convincing products. The design options and constraints have become exponentially more complex. While there are already a number of well proven, integrated tools to aid in the design of Graphical User Interfaces, the workbench for designers of physical interactions is still scattered.

Figure 1.5: Marble answering machine
By Durrell Bishop (drawing by Jonas Lowgren)

1.4 RESEARCH QUESTIONS AND GOALS

Based on these observations, this thesis embarks on answering the following questions:

- What is the process of designing interactive artifacts and what resources and tools does it involve? Design is guided by a carefully refined process. What is specific about designing physical interactions?

- Are there any problems with this process? What are the weaknesses, and how could this be related to insufficient tool support?

- What makes a good and successful design tool? Can we formulate general criteria that a good tool should obey?

- And finally, which specific tools can be designed to improve or enhance the design process?

1.5 OUTLINE

In an attempt to answer these questions, this thesis takes the following path: Based on the setting of the scene in this introductory chapter, chapter 2 will approach the use of tools analytically. The design process is taken as the foundation for an understanding of the broad range of activities that a designer is occupied with today. Then the set of available tools is surveyed with respect to the designer's needs and the strategies these tools employ to fulfill them. The chapter concludes with the discovery of gaps in the tool support of physical interaction design, towards the beginning and towards the end of the process.

Chapter 3 then distills the findings of the analysis into guidelines for tool creation. It additionally draws from relevant research and personal observations to suggest the three tool design pillars of creativity, craftsmanship,

and practicability.

An experimental attempt at filling the first of the identified gaps is made with the tool proposed in chapter 4. Sketchbook aims to support the designer in the early conceptual and sketching stages of the design. It integrates with the physical sketching activities by providing an accompanying structure and eased exploration. A sketchbook can be used informally, but it can also be gradually formalized to the point where it feeds directly into electronic prototyping toolkits.

The tool presented in chapter 5, Fritzing, aims to fill the second identified gap. It enables designers of physical interactions to move further towards high-fidelity creation of prototypes. It is designed to take over where the current prototyping toolkits stop, and provides an easy transition from the use of breadboards to professional production of PCBs. Through enabling documentation and sharing of designs for the first time, it can become a powerful driver of the design community. This chapter also recalls the guidelines established in chapter 3, and illustrates how they can be applied.

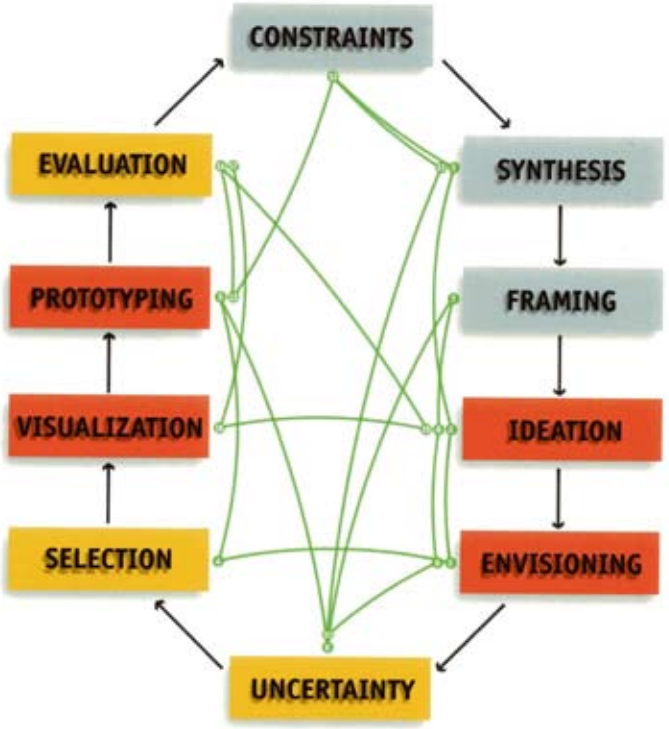
ANALYSIS 2

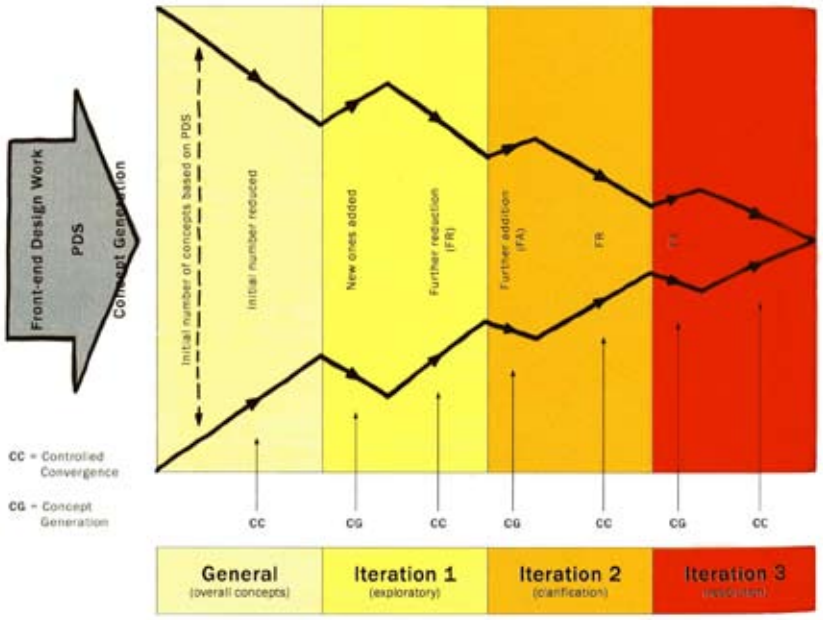
> It's a simple exercise -- a little logic, a little taste and the will to co-operate.
>
> Raymond Loewy about the design process

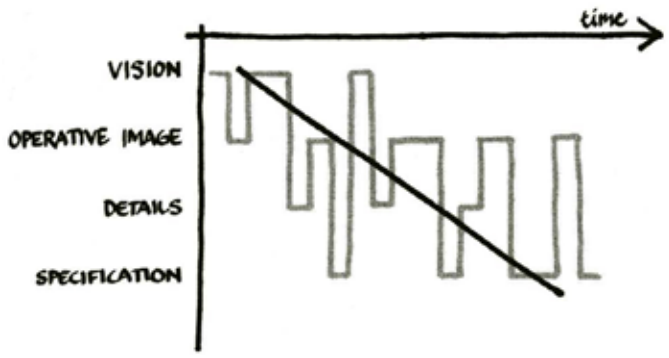
As we have already observed, we are using tools in a context: that of other tools (the tool set), a place (the workshop), and within a field of work. This chapter analyses this context of use by looking at the design process and the activities that constitute it, and relates them to the possible support with a specific tool set. A survey of existing computer tools in the domain of interaction design will highlight several approaches how these tools transform the design space to make it manipulable by the designer. Finally, this chapter identifies gaps that could benefit from additional support through tools.

2.1 MODELS OF THE DESIGN PROCESS

Design is generally structured along a process that usually starts with a brief and delivers a product, service, or other piece of work. There are as many variations to the process as there are designers, and design agencies advertise themselves with their perfected processes, but on a more abstract level it is always akin. The terms used to describe the individual elements differ depending on the perspective, but are often referring to the same thing. The British Design Council has just, in an extensive survey, analyzed the design process of eleven innovative major companies, and found "striking similarities" (Design Council, 2007, p.4), leading to their synthesized "double diamond" model (Figure 2.1d). These models are helpful in order to


CONSTRAINTS
EVALUATION
SYNTHESIS
PROTOTYPING
FRAMING
VISUALIZATION
IDEATION
SELECTION
ENVISIONING
UNCERTAINTY


Front-end Design Work
PDS
Concept Generation
Initial number of concepts based on PDS
Initial number reduced
New ones added
Further reduction (FR)
Further addition (FA)
FR
CC = Controlled Convergence
CG = Concept Generation
CC
CG
CC
CG
CC
CG
CC
General
(overall concepts)
Iteration 1
(exploratory)
Iteration 2
(clarification)
Iteration 3


time
VISION
OPERATIVE IMAGE
DETAILS
SPECIFICATION


discover
define
develop
deliver

understand more about some general requirements for design tools.

The diagrams in Figure 2.1 show four alternatives from different authors with different intentions. They all share the view that from far away the design process moves from a broad and open starting point through several focussing stages to the final concrete result. Buxton calls this the "design funnel", and as Moggridge, makes clear that the stages are in fact iterations. This suggests that tools must in the beginning be rather coarse and allow working with lots of ideas at very little opportunity cost. Further down the line the tools are staying similar in principle, but allow more and more refined work with more attention to detail.

Looking at the process more closely reveals an inherently erratic path, and while all diagrams acknowledge that, they describe different aspects of the apparent chaos: Moggridge, being more interested in the micro-steps of each stage, shows that designers in every situation make a new decision on what to do next. Löwgren & Stolterman notice a continuous jumping from the abstract to the concrete and back. This suggests that a tool should provide flexible entry and exit points, allowing the designer to use it opportunistically.

Buxton and the Design Council take explicit note of the divergence-convergence movements. This is also implicitly contained in Moggridge's diagram, where phases of ideation are followed by selection. From this we can deduce that a tool should support the easy creation and management of multiple concepts, as well as their comparison, in order to enable selection.

The design process thus differs strongly from a classical "waterfall"-like engineering process: Rather than trying to plan everything in advance and then following the plan step by step, it involves a lot of trial and error and testing of original beliefs. And this is exactly what the design process is tuned for, and what the tools have to support.

Figure 2.1 (left): Various descriptions of the design process
a) Top Left: Moggridge, 2006, p.730
b) Bottom Left: Loewgren & Stolterman, 2004, p.25
c) Top Right: Buxton, 2007, p.148, based on Pugh, 1990, p.75
d) Bottom Right: Design Council, 2007b, p.10

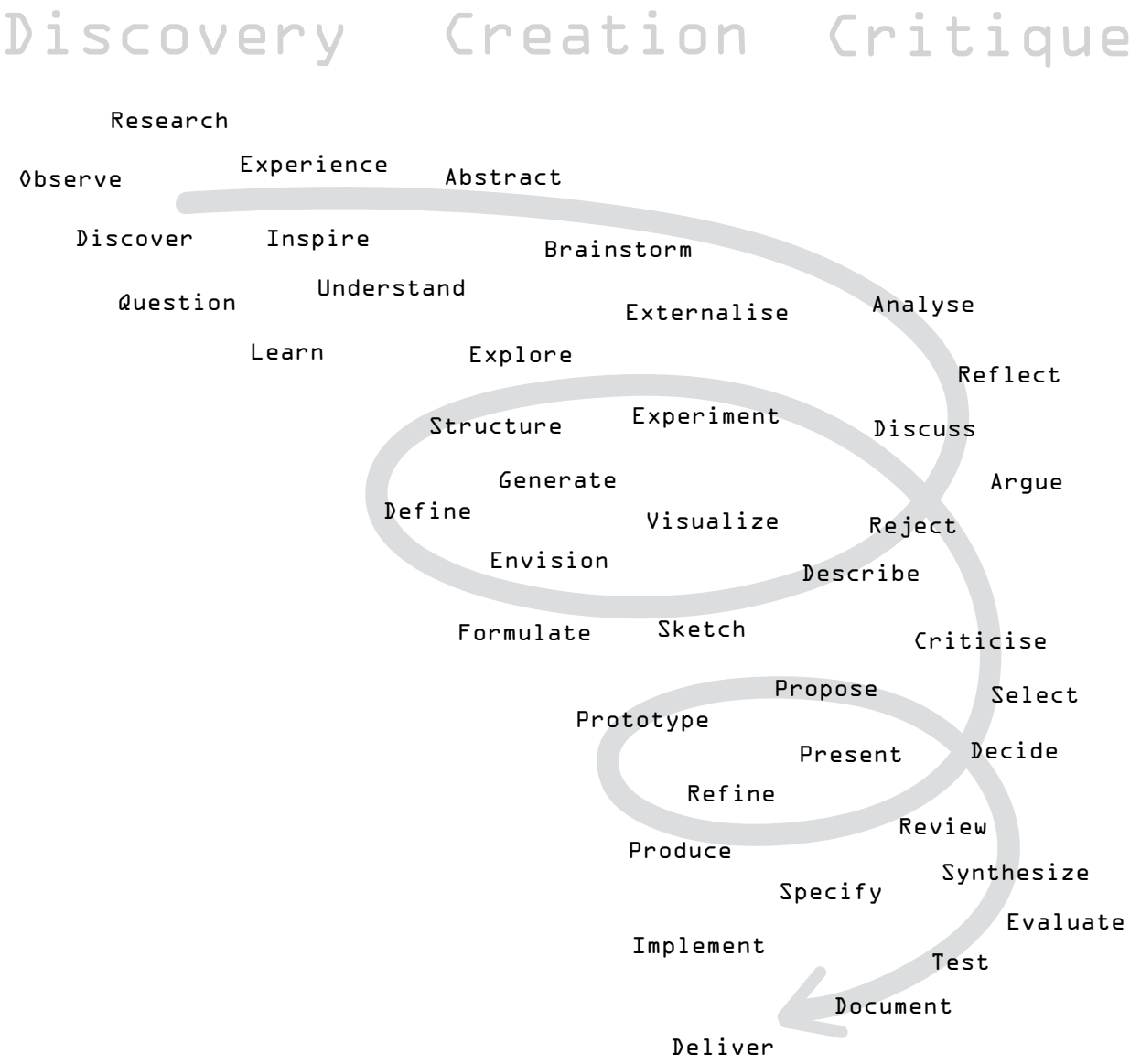


Figure 2.2: Map of design activities

2.2 DESIGN ACTIVITIES

In order to dive deeper into the design process and how tools are used within it, we will now look into the individual activities that constitute the designer's job. Activities describe on an abstract level what a designer is actually doing, and might be directly supportable with appropriate tools. Activities are the building blocks of the design process but due to its chaotic nature activities can only roughly be correlated to process stages. Figure 2.2 shows an attempt in naming relevant activities and clustering them: In the beginning they center around the discovery of the design space, then gradually move towards creation. In an iterative cycle between creation and critique the activities then lead to the final delivery. The activities mentioned here are neither meant to be non-overlapping nor exhaustive. Communication and collaboration are not separately mentioned as they can be regarded as part of many other activities. A few of them are examined here with respect to the employment of software tools:

Research

At the beginning of every project is the need of learning as much as possible about the design task and its context. Related projects, technologies, background theories, academic research, market opportunities, user opinions, as well as the client's situation are of interest and can serve as inspiration, opportunities or constraints. Löwgren & Stoltermann note that a designer's research is in principle different from academic research: While the latter is predominantly occupied with acquiring knowledge and truth, the former is using it as a means to provide a fertile foundation for design work (2004, p.31)[1].

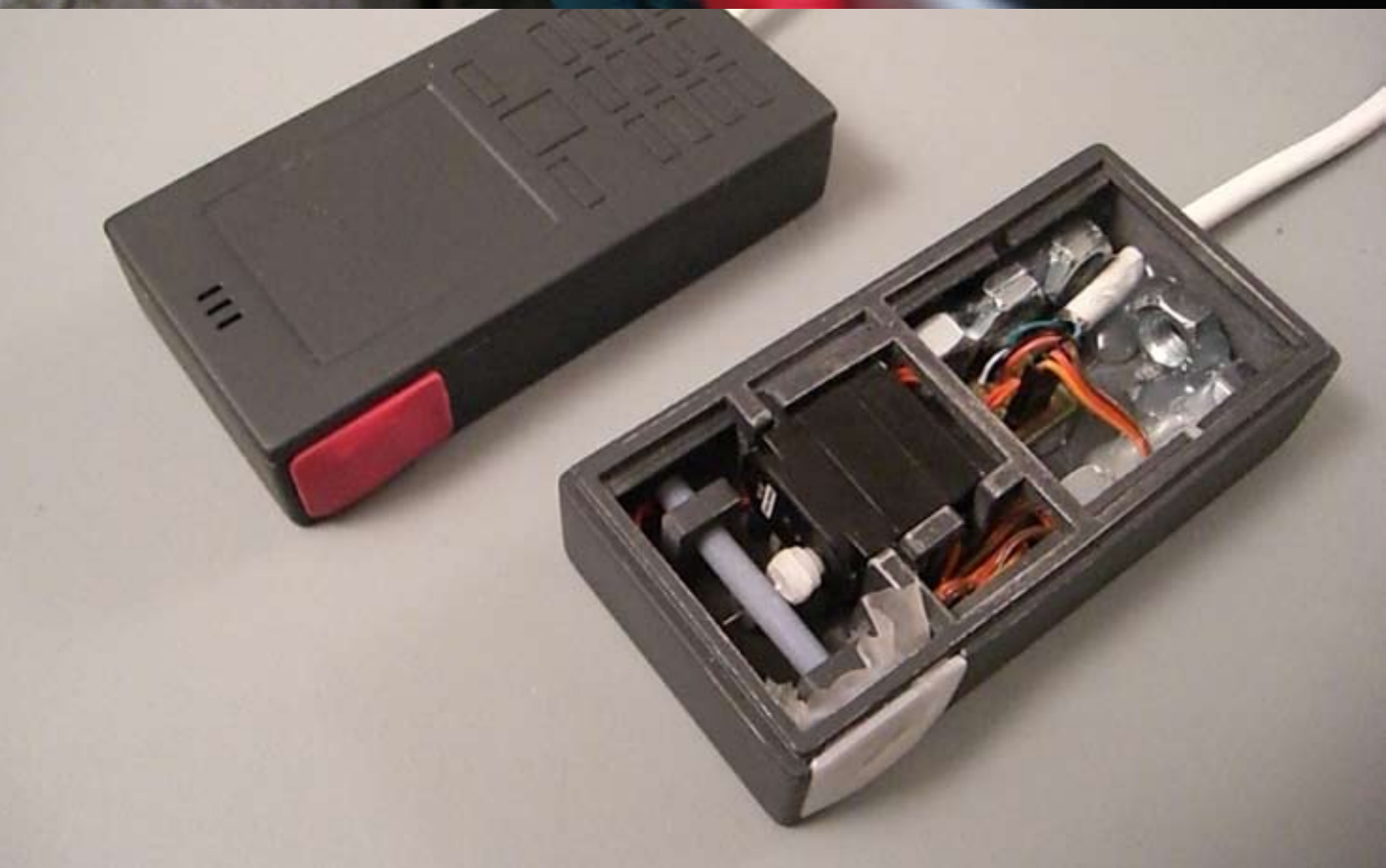
Tools can help with collecting research findings and make them accessible throughout the project. They can help to create an overview and structure the results in meaningful ways.

Abstract

The research results are usually specific to similar, yet other contexts. The designer therefore needs to abstract from these concrete findings in order to turn them into insights that could benefit his own design.

1 In German, these two notions are expressed with different words: "Forschung" refers to the academic type of research, whereas "Recherche" is closer to the designer's inquiry about the situation.


1 ao
4 ghi
7 pqrs



A tool could help to create diagrams of abstractions, while keeping references to the research that has led to them.

Envision

The vision plays a central role in the design process. It often starts out as a rough idea that is described with distinct words and visuals, sometimes metaphors. For the designer it is important to evolve this vision and continuously return to it as a point of reference.

Sketch

Sketching is the pivotal activity in the design process. Sketching is a cheap and effective means used to externalize concepts and ideas, present and discuss them. It allows the designer to quickly output a breadth of ideas, which as we have seen is crucial to initiating the design process. As discovered by Schön (1983), sketching is tightly integrated with reflection (the seeing-drawing-seeing-cycle) and therefore functions as a powerful catalyst: Through sketching an idea it automatically evolves.

Bill Buxton has written (2007) and lectured extensively on the importance and roles of sketching. He situates it explicitly in the beginning of the design process and distinguishes it from other acts of externalizing through its transient and non-committing nature. Figure 2.3 shows the continuum between sketch and prototype and figure 2.4 shows an example from the author (Hemmert et al, 2008).

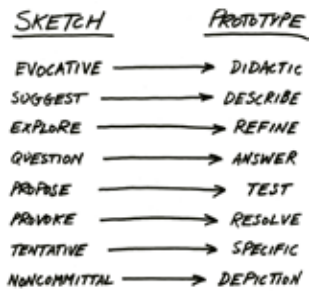


Figure 2.3: The Sketch to Prototype Continuum
Buxton, 2007, p. 140

A good sketching tool must be barely noticeable, fast and easy, yet at the same time highly expressive. It is hard to imagine a computer-based tool that is as ready-to-hand as pen and paper, but the computer could add the ability to make animated or even interactive sketches, more suitable for designing interactions. However, designers make very creative and opportunistic use of the material available to them for sketching their ideas. Besides all things pen and paper, any objects and materials, even electronics can end up in a bricolage to express an idea (Buchenau & Suri, 2000). The computer will therefore not necessarily be the central sketching tool.

Present

A designer constantly needs to present the current state of his work to his colleagues, the client, or other stakeholders. Sometimes it serves the

Figure 2.4 (left): Sketching vs. Prototyping
Different levels of fidelity

discussion of the work-in-progress, sometimes it needs to be highly polished, and designers are very aware of all the details that can influence a presentation's impact.

For review purposes design tools should allow a quick and direct presentation function for discussion, including the ability to annotate the presented designs. The requirements for external presentations are very different: They are created in a process just like the other design work and a tool should accommodate for this.

Select

"Design is a negative thing" says Buxton, and he means that design is about choosing the good ideas from the plentiful that are generated in the process. Selection happens all the time, when a sketch is rejected or it is refined. For larger decisions it follows a discussion and critique session, but it is the experience that allows the designer to make decisions quickly when not all suggestions can be fully explored.

A tool could help by presenting and comparing designs side-by-side and also record selection arguments. It could also trace the history of selections during the process, in the case that a selection needs to be defended again or a chosen path turns out to be unfruitful and needs to be reverted.

Prototype

Prototyping can be seen as an advanced, higher-fidelity form of sketching. Several decisions have already been made so that it is worth to put more effort into fewer prototypes. The prototype already gives a good experience of what the final product could be like.

Prototyping tools should allow an easy transition from earlier sketches, but can require more learning effort. They should be able to simulate an experience that is close to a final product and allow the designer to leverage cutting-edge technology without being an engineer.

Test

In many design disciplines, designs also need to be tested. Beyond technical functionality and reliability, the main test case is the user (or focus group) test. Does he understand it? Does he like it? Does he find it valuable, and also appealing? User testing is a tricky task because it is difficult to set up realistic testing situations and to minimize the observer effect. Also, the designer needs to be critical about individual user's opinions and abstract

from them, before he feeds it into his design.

Testing interaction design has become its own industry and profession, that of the usability engineers. As an engineering-oriented community, they have created an artillery of tools, methods and processes. Yet, designers need to do tests themselves, and therefore are in need of design tools that integrate testing functionalities. This requires the sketching and prototyping tools to be able to simulate interactive user sessions. Also, they could do minute recordings of test sessions and provide playback and reporting.

Produce

Nowadays designers more and more become producers of their own designs. This is a historic shift, because when the design profession was originally born in the age of industrialization, its main task was to merely create a model, the prototype, that would then be mass-manufactured by machines. The current evocation of a craft and DIY culture reflects this and even constitutes a democratization of innovation (von Hippel, 2005).

Because of readily available tools and machines, this trend is already a reality for most graphic designers. Scanners, color-proof workflows, layout and illustration software, and high-quality printers give them the ability to provide a full service, and even sell their own goods. The term DTP for desktop-publishing exists since the 1980s. When it comes to designing interactions, GUI designers are even better equipped. All machinery they need is a computer, and plenty of software exists that eases production by providing graphical design tools that more or less automatically translate designs into code (WYSIWYG: what you see is what you get).

Production is the classic domain of tool support. At this level tools need to give full control about every single detail of the design. A designer has to be knowledgeable about the involved production technology and the tool enables him to employ it to reach the highest quality.

Document

Finally, designs need to be documented on several levels. As already mentioned it is helpful to keep track of the decisions made during the design process, but whenever a design is delivered it needs to be accompanied by a documentation. It may contain usage instructions, construction and technical details, and contextual material that helps to see the design within a larger context.

Especially for technical documentation there is a lot of room for tool support. Designers need to communicate with engineers and other production people and the design tool should be able to export documentation in a suitable and compatible format. While this is often the case for production design tools, it would be helpful to start collaborating with engineers earlier on in the process. This suggests that also sketching tools could help to create formal descriptions of design sketches.

This quick overview showed the wide range of activities and tasks that designers are confronted with, and the potential and opportunities for tool support. Traditionally, software tools targeted at designers cover especially the 'produce' activity, stretching out into 'prototype' and 'document', but designers could benefit from tools in other critical activities.

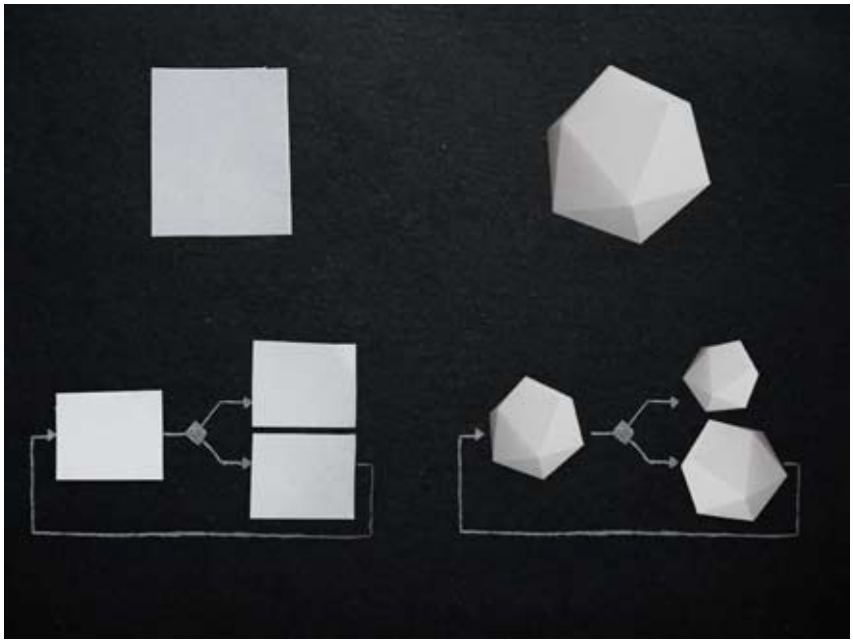


Figure 2.5: Interaction design
Graphic Design, Product Design, Graphical User Interface Design, Physical Interaction Design

2.3 METHODS & TECHNIQUES

A tool is not to be confused with a method: While tools assist in manipulating the physical reality, methods are descriptions of a procedure of how to approach or achieve something. Techniques are smaller entities of a method, usually related to a particular form of expression. For instance, the use of personas is an interaction design method where a prototypical user's biography and behavior are sketched out in order to approach the design in a more direct user-oriented way (Cooper, 1999).

Tools assist in carrying out methods, and often these tools are very small and specific. The Information Architecture Institute offers a set of tools of this kind on their web site[2]. For example, they offer templates for creating a task analysis and a diagram to perform the personas method.

A further interesting opportunity is in feeding more general design tools with integrated support for methods. This way, they encourage the use of best practice methods within a design process. A good example for this is the support for scenarios in rapid prototyping tools for web design.

2.4 DESIGNING PHYSICAL INTERACTIONS

The process and activities described so far are part of all design disciplines. But what makes designing physical interactions different? What are its specific challenges that a designer has to come to terms with?

From a design-historical perspective, the difference is in the unity of the skills of graphical user interface design and that of product design. The former were the first to experiment with the endless variations of interactivity offered by the computer, and the latter are masters of the object and also the electrically and mechanically induced interactions. Their combination is the Cartesian product of form and function: Any shape can have any functionality and interactivity (Figure 2.5). The design space is a 4D-medium where space and time can be freely designed, with few constraints. Therefore only a combination of tools, software and hardware, rather than one all-encompassing tool, can fully support this type of design.

The main design element is interactivity and this means creating a be-

2 http://www.iainstitute.org/en/learn/tools.php

havior that corresponds with a user. The designer is a choreographer of time and flow, of character and movement. It is obvious that a tool needs to help with exploring these interactive behaviors. It should give expressive control over time (states and transitions), behavior, and objects.

More than in any other design area, the context of use is of great importance. What is the situation, who are the persons, what are their intentions? These crucial aspects are often explored in character descriptions, scenarios and stories, much as in film. A tool could strive to integrate these with the design of the artifact itself.

In his philosophical treaty of interactivity, Dag Svanaes demands that interaction designers should learn to be good kinesthetic thinkers, for example by taking dance or Tai Chi classes (Svanaes, 1999, ch.12). They should be able to "think with their bodies" and have a trained sense of how people experience and interact with physical objects. For tools, Svanaes suggests that they should allow the designer to work in the same sense modality of the resulting product, not through abstractions: Design interactivity directly through interaction (ibid., p.231).

Finally, the designer needs to be acquainted with the technology that he is using in his product. For designing physical interactions, he needs to learn about high-tech electronics hardware such as controllers, sensors, and actuators. It is important to strike the right balance and to just learn about the opportunities and constraints of these elements, and not to get overburdened by technical details. Tools can provide exactly this: Expose the functionality and abstract the underlying technological complexity. Software tools have to integrate with hardware tools to enable the designer to build prototypes without the help of an engineer.

2.5 EXISTING TOOLS

The landscape of existing specialized tools is rather sparse, but there are a number of interesting tools from related disciplines. The following section highlights selected aspects that could be incorporated into tools for designing physical interactions.

Rapid prototyping for GUIs

Through its clearly defined constraints and standards, the design of graphical user interfaces for web sites or standard software lends itself per-

fectly to the support of complete prototyping suites. At least a dozen tools exist in this category, with several commercial versions, like iRise Studio[3] or Axure RP[4]. They all allow for quick WYSIWYG creation of GUIs, with different foci. The level of fidelity that can be achieved with some of them is close to a real application. For example, iRise offers the following specialised features:

Interaction flow editor (Figure 2.6) The designer can start by sketching just the abstract flow of interactions in a state-chart-like editor. Different paths through the application are composed by dragging and dropping pages and decisions, and connecting them. These can later be fleshed out in detail. Classical diagramming software like Microsoft's Visio[5] can also be used for this, and there is even a set of specialized diagram stencils available, the "visual vocabulary"[6]. These tools are however not providing the benefits of integration.

A disadvantage with this tool is that it lures the designer to think of the interaction as a sequence of states/pages, which as a general concept is problematic. With technologies like AJAX and Flash, individual pages become more and more dynamic and offer rich interactions, which can no longer be modeled in such a diagram. Still, they help to keep an overview and discover structural problems and gaps.

Annotation during presentation (Figure 2.7) The integrated presentation functionality is enhanced with the ability to add comments and directly link them to items in the current view. These notes can then be accessed in the design view and reviewed.

Integrated documentation (Figure 2.8) A classical report-like documentation is automatically assembled from the design. This is again tightly integrated with the design, with comments visually linking to page elements. The documentation serves as a complete description for implementation.

Sketchy rendering

A lower rendering quality is often chosen intentionally by designers in order to make clear that the presented sketch is unfinished. People asked to criticize it are then more apt to do so. The perfect look of computer generated drawings, even if sketches, thwarts this. Some tools therefore employ

3 http://www.irise.com
4 http://www.axure.com
5 http://office.microsoft.com/visio
6 http://www.jjg.net/ia/visvocab

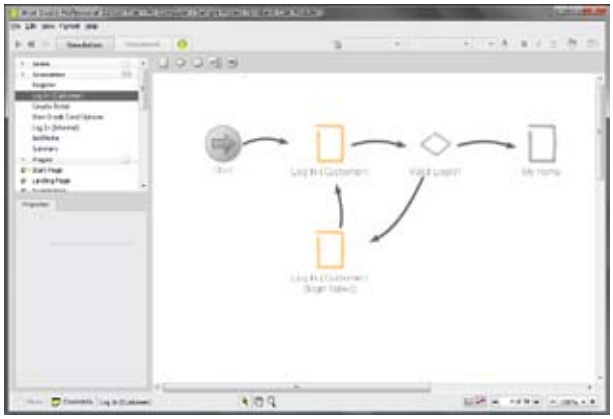


Figure 2.6: Interaction flow editor
iRise Studio

Figure 2.7: Annotation during presentation
iRise Studio

Figure 2.8: Integrated documentation
iRise Studio

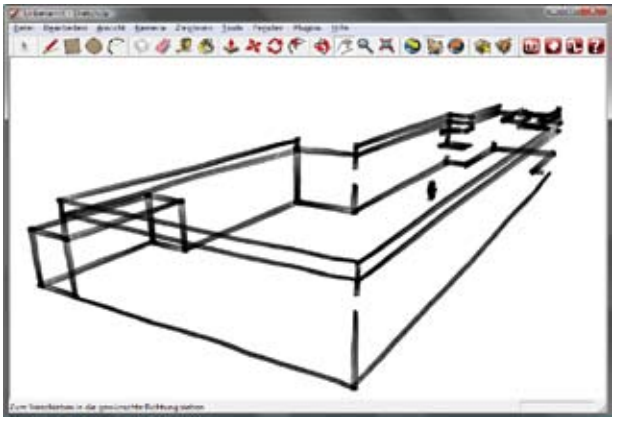


Figure 2.9: Sketchy rendering
Google SketchUp

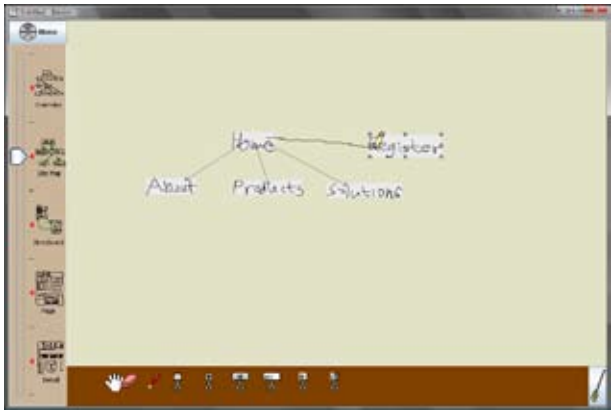


Figure 2.10: Sketching
DENIM

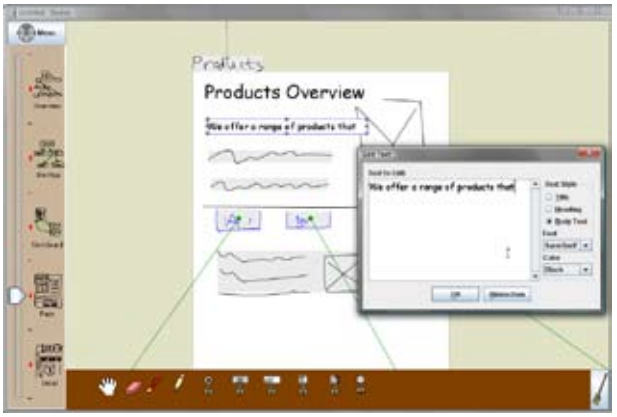


Figure 2.11: Progressive refinement
DENIM

algorithms that give their renderings a hand-crafted look. Figure 2.9 shows a Google Sketchup[7] rendering of a building in a marker pen style.

Early-stage sketching

DENIM[8] is a sketching tool for the early stages of web site design. The authors identified the gap between sketching with pen on paper and only later transferring the initial ideas to a software tool for refinement. Their solution lets the designer use the computer right from the start. Being a research project, DENIM makes use of several unusual interface ideas. (Lin et al., 2000)

Sketching (Figure 2.10) The software is to be used with a pen tablet and makes consistent use of this fact. With the pen tool pages, connections and texts can be drawn directly and are automatically recognized, and the tool can be switched to a hand, an eraser, or stamps for placing widgets. The sketchy look is retained. Advanced users can learn shortcut pen gestures and use a pie menu for additional functions.

Integrated multiple perspectives Sketching a rough outline, a site map, some storyboards, and also page designs–the designer jumps seamlessly from one to the other, leaving unclear or refining whatever is opportune. Since these different perspectives effectively represent different levels of abstraction on the same content, a zoom slider can be used to move back and force between them (Figure 2.12). Finally, the system also provides a (separate) interactive presentation mode.

Progressive refinement (Figure 2.11) Inherent in the levels of abstraction is the possibility to move from a coarse design in the beginning (e.g., the site map) to finer levels later (e.g., sketching page details). Additionally, pen drawings can progressively be replaced with higher-fidelity computer drawings, while retaining the syntax.

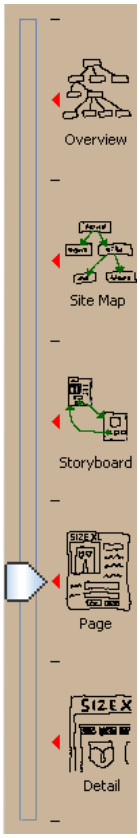


Figure 2.12: Zoom levels DENIM

Exploring interactivity

Providing a way to experiment with interactivity effectively means giving an expressive access to programming. Several fundamentally different approaches exist, with different trade-offs.

Design-oriented API (Figure 2.13) The simplest way to make programming interactivity accessible for designers is to provide a simplified, de-

7 http://sketchup.google.com
8 http://dub.washington.edu:2007/denim

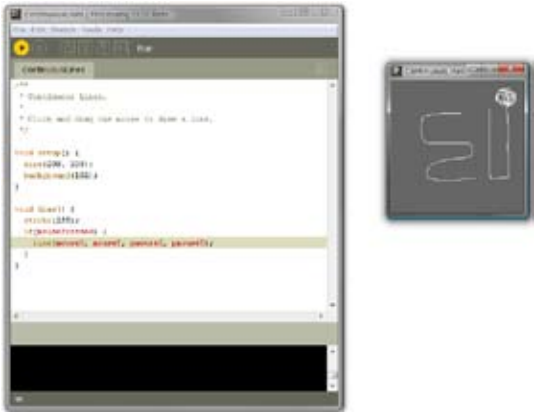


Figure 2.13: Design-oriented API
Processing

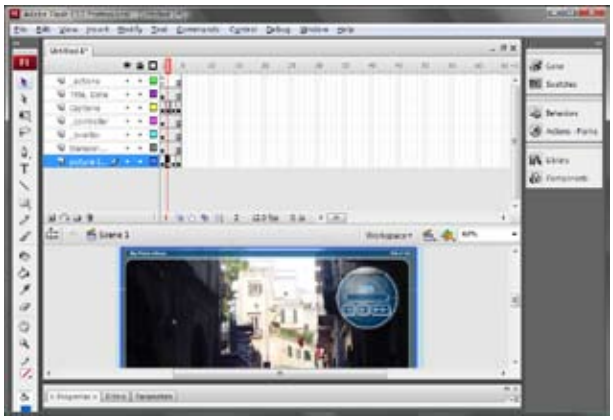


Figure 2.14: Timeline
Adobe Flash

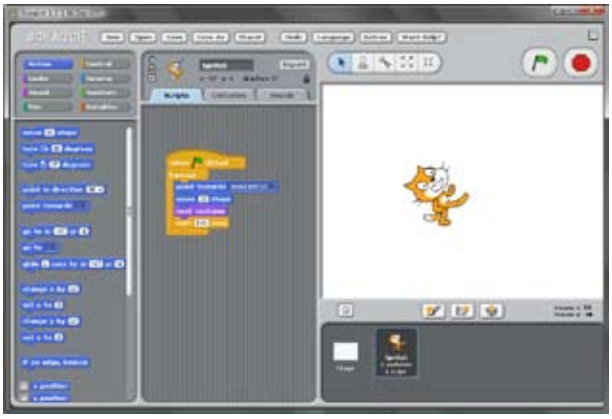


Figure 2.15: Stage, actor, behavior
Scratch

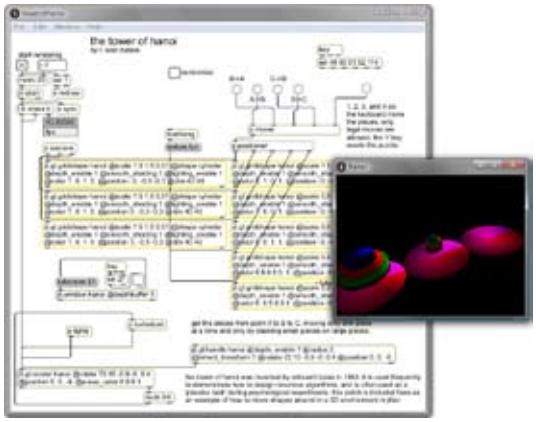


Figure 2.16: Visual progr. language
Cycling74 Max

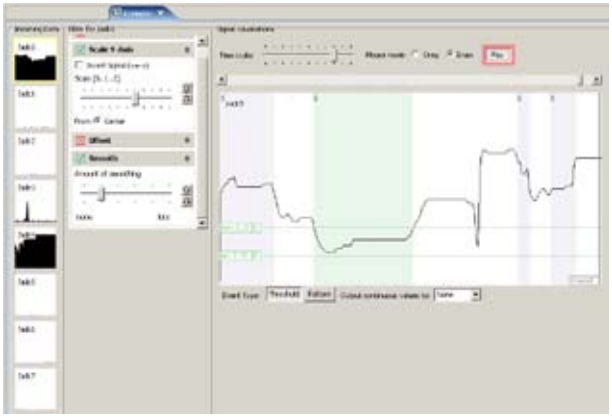


Figure 2.17: Progr. by demonstration
Exemplar

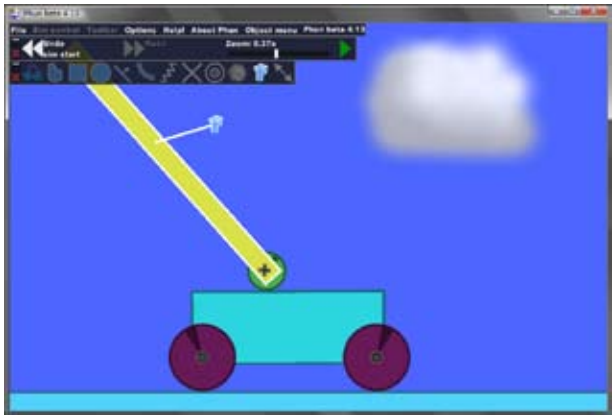


Figure 2.18: Physical simulation
Phun

sign-domain-oriented Application Programming Interface (API). This is part of the success of the Processing environment[9] (Reas & Fry, 2007), an open-source programming environment for designers and artists. Another aspect here is the extremely low entry barrier that hides all of the complexities usually involved with programming, while letting the user move on far beyond the foundation provided by the tool (low threshold, high ceiling).

Timeline (Figure 2.14) Interactivity is time-based, and so several authoring environments make use of the easily understood concept of the timeline. It lends itself especially to creating frame-by-frame-like interactions such as presentations or menus. Adobe's Flash[10] has a highly evolved timeline with layers and different types of frames, and combines it with scripting to overcome its limitations.

Stage, actor, behavior (Figure 2.15) Because of its real-life-metaphor, programming by placing actors on a stage and assigning them behaviors is an even easier graspable concept. Because it also mimics the modern object-oriented programming style, several educational tools like Scratch[11] and Alice[12] use this approach. They both combine it with a refined visual approach to compose behavior.

Visual programming languageIn order to avoid the need to learn the syntax and semantics of a programming language, several tools use a more "tactile" approach. The user can connect functional blocks to create an algorithm. The various languages range from tools with rather low-level blocks for professional media production like Max[13] (Figure 2.16), PureData[14], or vvvv[15], to higher-level block languages in educational tools like Scratch (Figure 2.15), Alice, or AgentSheets[16]. The latter enforce a higher degree of structure and abstraction for the benefit of learning. Visual programming languages are very popular among designers, not just because of the fact that they are visual, but also because they tend to deliver results quickly. Their major disadvantage is that they quickly become incomprehensible with larger projects.

Programming by demonstration The principle behind this is to perform

9 http://www.processing.org/reference
10 http://www.adobe.com/products/flash
11 http://scratch.mit.edu
12 http://www.alice.org
13 http://www.cycling74.com/products/max
14 http://www.puredata.org
15 http://www.vvvv.org
16 http://www.agentsheets.com

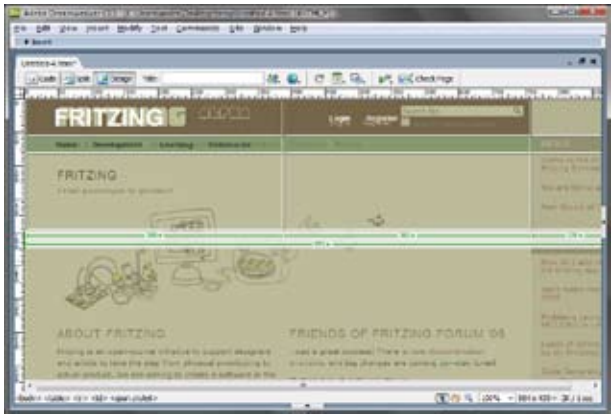


Figure 2.19: Transparent blueprint
Adobe Dreamweaver

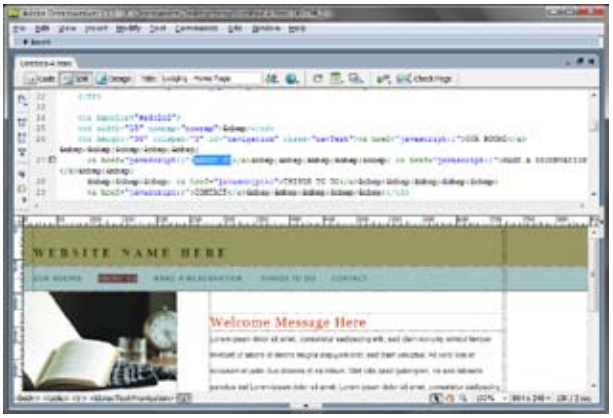


Figure 2.20: Design view vs. implementation view
Adobe Dreamweaver

Figure 2.21: Input-output box
Arduino

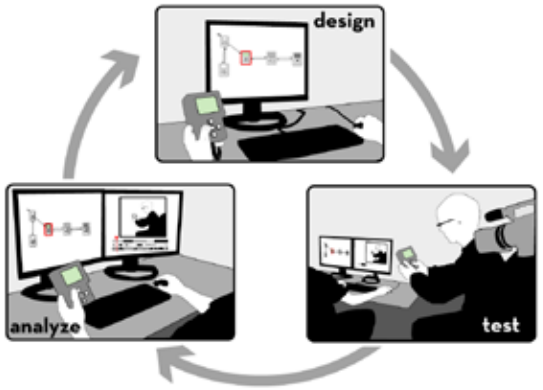


Figure 2.22: Integrated prototyping
d.tools (Workflow)

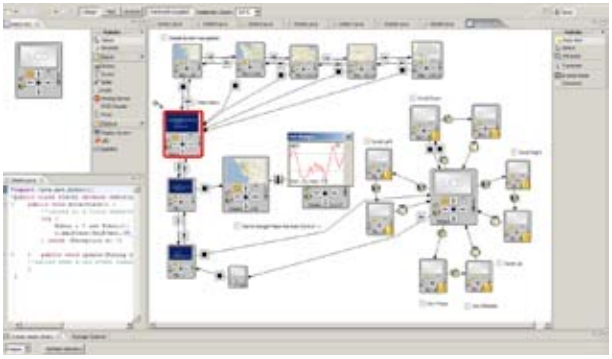


Figure 2.23: Integrated prototyping
d.tools (Design view)

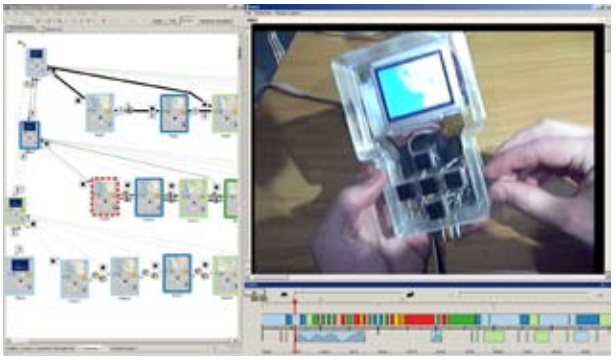


Figure 2.24: Integrated prototyping
d.tools (Analysis view)

the intended behavior of an object by, e.g., making a mouse movement. The computer will then record this behavior and assign it to the object. This method has not been very successful yet. Several research projects in interaction design it for creating graphical animations (K-Sketch[17], Davis et al. 2008) or for sketching continuous interactions (Li & Landay 2005). When designing physical interactions, such an approach can be very helpful to design sensor-based gestures. The Exemplar toolkit[18] lets the designer demonstrate a behavior with the actual sensor and provides means for editing and utilize the resulting values (Hartmann et al., 2007, figure 2.17).

Physical simulation (Figure 2.18) A form of expressing interactivity that could be interesting for some types of physical interactions is through physical simulation. Tools like ASSIST[19] (Alvarado 2000), or more recently, Phun[20], allow sketching of 2d objects in a world that obeys physical laws. Different mechanical elements like hinges and springs, as well as material properties can be simulated. However, these tools only allow effective 2d simulation at the moment.

In general, it can be said that while all these alternatives are intuitive solutions for particular problem sets, an actual programming language still allows for the largest expressivity.

Production

For some domains, the tools have matured so far that they enable the designer to create the the actual product all by himself. Especially in the domain of web design, the availability of such tools have spawned an enormous amount of self-proclaimed full-service designers. For instance, Adobe Dreamweaver[21] is a highly evolved tool that lets the designer incorporate a broad range of modern front- and back-end technologies. At this stage, the designer needs to be somewhat intimate with the technology, as every adjustment is directly translated into code, but the use of assistants, examples, and tutorials ease the learning curve.

Transparent blueprint (Figure 2.19) Since the wysywig editor here does not allow as much freedom as in prototyping or graphic design tools, precreateddesigns can be embedded as a semi-transparent background. The

17 http://www.k-sketch.org
18 http://hci.stanford.edu/exemplar
19 http://rationale.csail.mit.edu/project_assist.shtml
20 http://www.phunland.com
21 http://www.adobe.com/products/dreamweaver

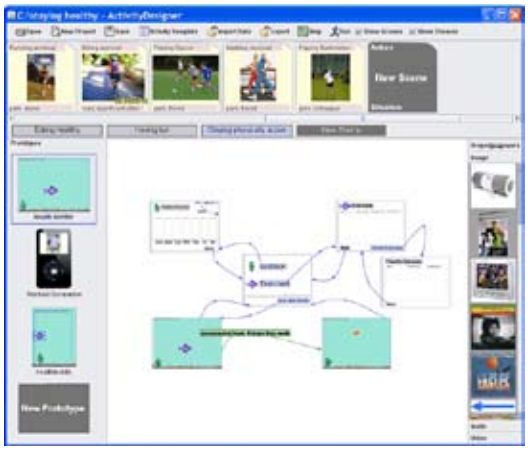


Figure 2.25: Activity-centered design
ActivityStudio

designer can then use the production tool to recreate the design as closely as possible.

Design view vs. implementation view (Figure 2.20) It is essential for the designer to stay in full control of the technical implementation, because every detail can have an effect on the outcome. It would be deceptive to just work with the wysiwyg view. Because of that, production tools often feature prominent functions to switch between a design and an implementation view.

Design tools for PUIs

The field of physical interaction design is relatively young, but has seen a steep increase of interest among designers. This is not least due to the recent availability of easily accessible electronics prototyping toolkits. Apart from that, however, there exist hardly any tools.

Input-output-boxes The aforementioned prototyping toolkits are usually in the form of "input-output-boxes", because this model coincides well with a naive view of an embedded computer. These are easily programmable micro-controllers that can read from digital and analog inputs and write to respective outputs. Originally targeted at beginners of learning electronics, they were usually either too limited or too complicated to use. The Arduino platform [22] (Mellis et al., 2007) was the first to effectively target designers and artists and brought a breakthrough. It combines a robust microcontroller with a simple development software (Figure 2.21). The technical know-how required to hook up electronics is kept at a minimum.

Integrated prototyping A first step towards a more fully integrated toolkit was taken with the research project d.tools[23] (Hartmann et al., 2006). It combines design with testing and analyzing (Figure 2.22). The main principle is a close and automatic coupling between electronic building blocks and their visual representations in the software. Based on this, a graphical state-chart editor lets the designer visually program the artifact's behavior (Figure 2.23). Once the design is ready, it can be tested with the physical device. During the test the software records an event log and synchronizes its with a video recording. This allows for convenient and integrated analy-

22 http://www.arduino.cc
23 http://hci.stanford.edu/dtools

sis of the user tests which can even be navigated by the physical device (Figure 2.24).

The tight integration furthers frequent testing and fast iterations, while focusing on the interaction design. A disadvantage is the requirement of specialized hardware which makes the electronics somewhat inflexible and clunky, The system is therefore mostly used for medium-stage sketching.

Activity-centered design (Figure 2.25) A unique approach to designing for ubiquitous computing support of long-term activities is proposed in ActivityStudio[24] (Li et al., 2008). The toolkit provides a similarly integrated environment as d.tools, but adds a focus on users' activities and narratives. Activities are treated as a first-class design object and together with storyboards build the basis of a design prototype. ActivityStudio also proves that it is possible to build a design tool based on a general theory of interaction design ("activity theory", Kaptelinin & Nardy 2006).

2.6 GAPS

The investigation so far has provided a map of the design process and relevant activities, and the overview of existing tools has shown how interaction design is currently covered with support through computer tools. The set of tools is quite rich and manifold, but mostly with respect to the design of graphical user interfaces. Figure 2.26 maps the main tools [25] of the core design activity (i.e., designing) against the main process stages.

Graphical interaction design is very well supported along the whole process, with several alternatives for each stage. Only in the early concept stage there is just a research project available – due to the moderate complexity involved and effective concept work with pen and paper it can be argued that the need is limited here. For designing less standardized interactions, another set of tools supports sketching and prototyping well, but unless the designer is also an expert programmer, it comes short of professional, distributable production.

The support for physical interaction design is even more confined to sketching and prototyping. Arduino is a classical prototyping toolkit, while d.tools is more directed towards sketching. Unfortunately, like DENIM,

24 http://activitystudio.sourceforge.net

25 In the case of similar tools the most known one was picked as a representative.

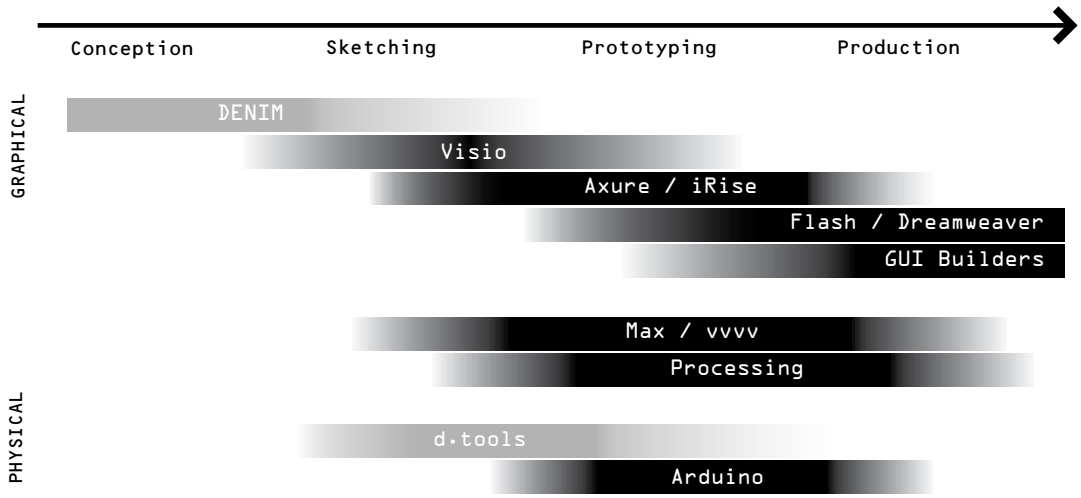


Figure 2.26: Matrix of interaction design tools
(Software tools only; research projects in light gray)

d.tools is a research project that is no longer actively developed. The gaps are easily identified at both ends of the process: Concept work as well as production are not currently supported by tools.

But what does that mean? Are these tools necessary? Concept work is difficult to support because of the complex and essentially open design space. The first stages are explored with hand sketches and materials in the real world. However, these are usually not interactive and difficult to maintain, and it could be helpful to have a tool that integrates into current practices to improve this situation.

For the far end of the process, when the design nears a production level, designers currently need the assistance of electrical or mechanical engineers. They can build fully interactive artifacts, but they remain far from a redistributable form. It could be argued that designers simply lack the engineering knowledge required, but the GUI design tools have successfully shown how this knowledge can be leveraged for designers. Here lies another opportunity for empowering tool support.

Before these two opportunities will be explored in detail, the next chapter will be occupied with the tools themselves and extract guidelines for what makes a powerful tool.

DESIGNING TOOLS

3

A good tool is one that is widely used and is effective and efficient towards its purpose.

I made up this definition by hi-jacking that of creativity, similar in spirit: "Creativity is the ability to come up with ideas or artifacts that are new, surprising and valuable." (Boden, 2004, p. 1). It could be a definition of what makes a good tool, and still not explain what that is like, or how it is made. Nevertheless it is a sensible definition, because it is led by practicality, measuring the success of a tool by the overall impact. 'Widely used' is meant with respect to the potential number of use cases, and to counter-balance cases of forced monoculture, two measures of usefulness are added. A good tool is used effectively, i.e., its user can achieve what he wants, and he can do so efficiently, i.e., with a minimum use of resources.

But we will here be concerned with how to create a tool that reaches this state. The previous chapter offered concrete opportunities for tools. This chapter collects a set of criteria that should guide the tool-maker and focuses on three main topics that are especially relevant for making design tools: creativity, craftsmanship, and practicality.

3.1 CREATIVITY

Any design tool should support creativity. The work of the designer nowadays is centered around this word, and as we saw, the design process is

tuned towards a creative result. The research into "creativity support tools" has recently seen an intensification and there are now several recommendations on how to construct a tool, summarised in Resnick et al. (2005).

Exploration and experimentation

The tool should support an exploratory approach and facilitate experimentation. This is contrary to a waterfall-like tool that lets the user move step by step towards the final outcome. Designers want to be aware of all the expressive freedom that a tool provides and safely and effectively compare alternatives of how to proceed.

This requires the tool to be trustworthy. The user should always be in charge and aware about the state of the program. Automatic saving and infinite undo are basic functionalities to ensure a safe environment, but it can be enhanced with a thoughtful and detailed interaction design that lets the user see and anticipate the effects of his actions.

Furthermore, the tool needs to reveal its scope and possibilities. Ideally, it does so in an incremental fashion, so that the user can grow with his skills and is not overburdened from the start. Besides a thorough documentation, a very effective way is to provide a broad range of practical examples. Designers are highly inclined to look at these, experiment with them and see the effects – trial and error. This should not be underestimated, especially since it takes a great effort to create good examples. Other techniques for revealing are auto-completion or visual browsers.

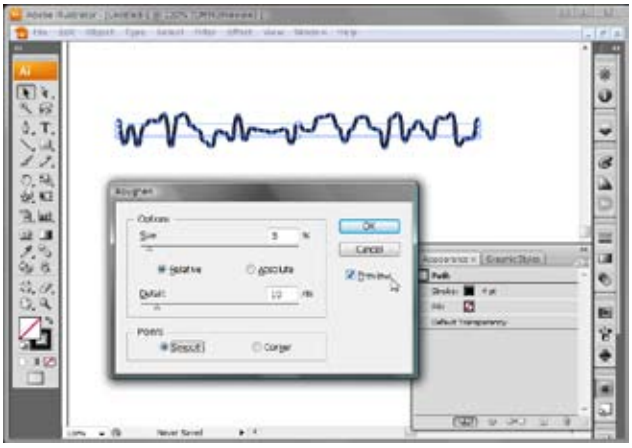


Figure 3.1: Preview and object styles
Adobe Illustrator

The need to experiment can be explicitly addressed by supporting variation and alternatives (Terry et al., 2004). E.g., instead of a monolithic project file, the tool could treat a project more like a tree with the ability to branch from one version into multiple alternatives (much like a code version control system). Similarly, variations could be supported on a per-document scale (e.g., by different style sheets). The Adobe tool suite offers per-object styles that can be safely explored with a preview and modified later (Figure 3.1).

Low threshold, high ceiling, wide walls

This metaphorical description of the room that the tool architect should construct refers to the learning curve. First of all, the tool should be very easy to get started with (low threshold). From opening it up it should be immediately possible to be expressive, without the need of making any configurations up-front or having to go through a lengthy compile process to view the result.

Figure 3.2: Integrated online sharing Scratch

Even though the start should be simple, the tool should not limit the user in his expression. The tool should be open-ended and allow professionals to work on sophisticated projects (high ceiling). This need can sometimes conflict with the first, when the ease of making simple things prohibits the capability to make complex things. However, if the tool is targeted at an early design stage, a low threshold is more important, whereas a high ceiling becomes more important in a tool made for a later stage.

Finally, the wide walls suggest an openness of the design space. The tool should not be too specific in the range of things that can be created with it. Otherwise, all results will be similar and the room is too narrow for transformational creativity to occur. A design tool should not be limited to certain use cases and a set of predefined modules. It should be inherently open and leave it to the user what he intends to create, and also how he wants to create it.

Informality

Another strategy to support the erratic design process is to allow for an informal tool use (Cook & Bailey, 2005). While pen and paper support this type of use very well (you can sketch anything you like in any degree of refinement), the computer is the epitome of formalism. This is exemplified in many tools, where only predefined actions can be made. While it is not possible to overcome this per se, it can be diminished by allowing

free-form annotations and relaxing composition rules. For example, a GUI design tool could allow the temporary placement of widgets in free space before requiring a layout rule. This supports the designer in beginning a composition without committing to the details yet, and lets him off-load ideas through external representation.

Collaboration and community

As already discussed in chapter 2, collaboration goes beyond the shared access to project files: Since there is usually only one designer in charge of a project part, it is more important to have a good presentation functionality, ideally combined with an annotation facility.

Csikszentmihalyi has shown that creativity, even though it starts from the individual, is a social process (1996). It involves the leaders of the field of practice and the foundation of the domain. The recent upsurge of a creative do-it-yourself community proves that technology, especially the internet, is a key factor for fostering creative communities. They revolve around web sites like instructables.com, makezine.com, or etsy.com.

Design tools can build their own online communities to multiply the creative outcome that can be produced with the tool. Processing[1] has been the first to be highly successful with this approach by making sharing an essential part of the tool, and Scratch[2] has evolved this: The software contains a prominent one-click "Share it online"-button, and projects from the web gallery can be downloaded and opened with one click, fully editable (Figure 3.2).

3.2 CRAFTSMANSHIP

Design has originally evolved from craft, and many of its values are popular today. Tinkering with the materials, and only allowing the highest quality, the craftsman strives to unite utility and beauty. He builds up his tool set with great care, adds one by one as his skills grow, and learning more and more about the character of each. As the craftsman grows to become a master, he begins to customize his tools, making his own to suit his working style and to achieve what was not possible, or not as comfortable, with

1 http://www.processing.org
2 http://scratch.mit.edu

the others. This image of the craftsman as a master tool user should guide the tool maker.

Detailed control

A software design tool often uses abstracted representations of the design object to enable an eased, symbolic interaction. This should however not come at the cost of fine-grained control, especially at the later stages of design, when the designer cares about every tiny aspect of the product. Simplification should at this stage be valued less than providing absolute control – acquiring mastery and honing ones skills is a necessity of becoming a good designer. One has to get intimate with the design material, and in the case of the interaction designer this is computer technology. A good example is the simultaneous view of design and code in Dreamweaver (Figure 2.20). For earlier stages, the control may be less fine, just as the craftsman may use a coarser tool to give the object its first rough shape.

Figure 3.3: Custom-made tools of a ceramic craftsman
Photo by Cory Lum

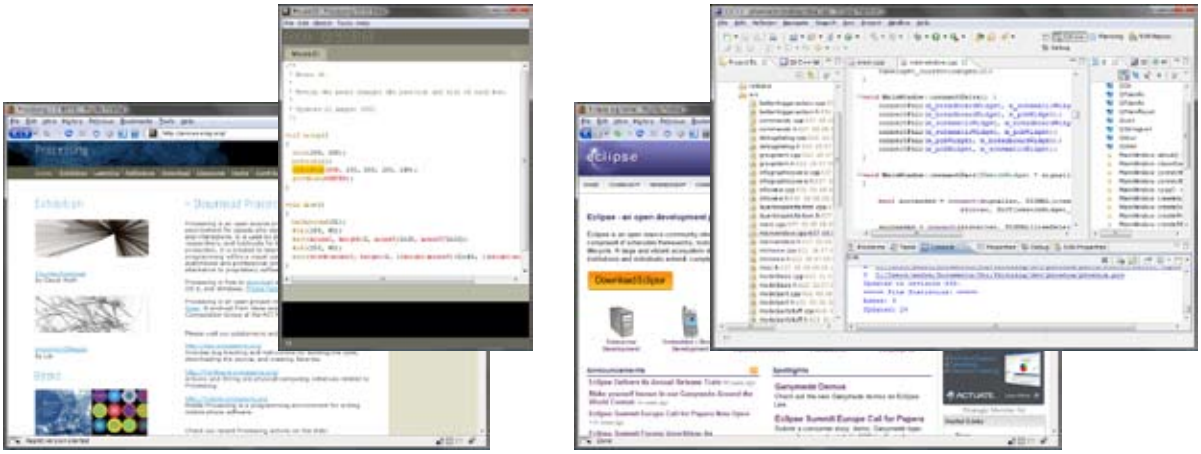


Figure 3.4: Tool aesthetics of different communities
Processing vs. Eclipse (web site and software)

Tool appropriation

As we saw, as the skills in using a tool grows, the advanced user seeks to appropriate the tool to make it a tighter fit with himself. The tool becomes transparent to the master user, they become a unity to the amazed observer. Accustoming for the needs of different skill levels, from novice to expert, is a classic in user interface design, but often only means custom settings, shortcuts and macros. Designers might want to customize a tool even further by changing certain behaviors, or extend it by adding functionality and modules (Figure 3.3). Open-source tools obviously make this possible, but only for expert programmers. A more accessible option is the integration of end-user programming like a scripting language.

Focus

A tool should do (at least) one thing very well. It does not help the designer if a tool can do many things more or less well, if the core purpose is not handled perfectly. The tool maker should therefore focus his work on this core functionality and not divert it into additional features before one is done right. Focus makes a tool valuable, because it gives the designer a reason to use a particular tool for a certain task: it is the best. Additionally, the tool's user interface will stay simple in the best sense, focused on the main activity without diversions. More functionality that is added later naturally receives lower priority in the interface.

The downside of specialization is the time involved to learn and switch between many different tools. This can be balanced by compatibility between tools, common formats and interfaces (see also "Flexibility").

Aesthetics

Not only should a tool enable its user to create aesthetic artifacts, it should also have an appealing aesthetic to itself. Designers are critical about aesthetic properties and are more likely to use and identify with a tool that pleases their eyes and has a good feel. Aesthetics add to the joy of use, but it is also the trust that only an aesthetic tool can be a good tool. The look-and-feel should reflect the user group and their approach. For example, compare the look of the Processing tool and web site to that of Eclipse (Figure 3.4). Both tools essentially provide the same functionality and are very popular, but while Eclipse is made for engineers who like features, Processing clearly targets designers who prefer simplicity and focus, and feel more at home in this environment.

3.3 PRACTICALITY

All of the above is worth nothing if the tool is not used because it ignores the designer's day-to-day requirements of the job. In their survey, Myers et al. identify a number of promising research concepts for design tools that have never caught on in the community (Myers et al., 2000). They were too hard to learn, required the designer to follow a prescribed model too narrowly, or simply were not paying off the effort. Stolterman et al. (2008) ascribe this to a missing understanding of design practice among researchers who are proposing new tools. Tools have to prove their practicality through use, and it is important to design a tool so that it survives at the hectic designer's workplace.

Cost

The decision of adopting a tool is obviously influenced by its costs. Designers tend to choose their main tools carefully because they will become heavily invested in it to build up skill and expertise. The main set of tools is usually comprised of only a few, and they can be more costly in terms of price and learning effort. A tool that functions as an additional utility should have the opposite characteristics. In general, designers are often

pragmatists and result-oriented (one could also say effect-oriented). Resources are precious and used to the greatest effect. It does not matter so much how something is done, if the result is satisfactory.

Of course the financial cost should in general be low to allow wide-spread adoption, but a good tool that saves time or enables the designer to accomplish something that is not possible otherwise can ask for a higher price. However, as open-source tools are gaining market strength, the customer's expectations are changing and tool makers have to look for alternative ways to collect money. E.g., the Arduino team is earning its worth through selling the hardware and offering workshops and consultancy. Also big companies such as Adobe are beginning to offer their tools as open-source (e.g., the Flex SDK[3]) to ensure the adoption of their technology – which can be best designed for with their commercial products.

The other important cost factor is time. If a tool is considered one of the core tools of the designer, an extensive learning phase at the beginning is actually acceptable. The high threshold here comes at the benefit of a high ceiling, i.e., a powerful and expressive tool that can be applied universally. Also, the tool should then be very efficient, costing little time to achieve a lot. More time can be gained by providing open interfaces to other tools. Compatible file formats alleviate the cost of moving from one tool to the other.

Flexibility

A high investment in a tool can only be made if the tool is very flexible. It should allow a wide range of applications, and be customizable and extensible by the user or third parties. The tool maker should announce a roadmap that further extends this flexibility. Instead of trying to bind his customers through proprietary standards, he should make use of open standards and open interfaces. Again, Adobe is a good example of a company which has successively opened its base technologies (Flash) when customers were already on the fringe to adopt other technologies (AJAX). Besides eased interchange with collaborators, the use of open file formats allows the designer to work on the files with other tools which might be more specialized for certain things, or even manually.

3 http://opensource.adobe.com

Periphery

The analysis has shown that the designer's work also consists of a lot of important peripheral activities. Even though the tool should focus on what it does best, the tool maker should be aware of the context of use. The tool has to function in day to day's work and not present a stumbling block when the designer needs to transition from one activity to another.

For example, the tool should allow easy sharing with other stakeholders without lengthy instructions and without the need of a licensed copy of the tool. It should have simple means to directly present the design, without distracting user interface elements. It should allow the designer to include notes and documentation, for others and for himself. At the end of the process, the work should be archivable in a way that is retrievable even after years.

3.4 CONCLUSION

As to be expected, there is not the one set of criteria that defines a good tool and therefore the guidelines are rather broad. The tool maker must be very aware of the purpose of the tool and the role it will play in the designer's work. What is the main focus of the tool and which problem does it solve? Is it a core tool or just a utility? Is it a composition or an exploration tool? Which amount of learning can be justified? And how can it fit into the landscape of existing tools?

These questions need to be assessed carefully and the tool designed accordingly. The next chapters will present two tools that will be approached in this manner. Chapter 4 describes a concept of an informal sketching tool, and chapter 5 describes a practical technology tool that is actively being developed.

TOOL I: SKETCHBOOK

4

This chapter presents a concept for a tool named Sketchbook. It is an answer to the first gap discovered in tool support for physical interaction designers. Sketchbook aims to support designer in the early conceptual and sketching stages by providing structure and simple interactivity to the chaos of physical sketching work.

As opposed to the real-world use case in chapter 5, the approach here is experimental. The tool design is not restricted by real budget or technological constraints, to give room to new interface design ideas and an exploration of the guidelines established in the previous chapter.

4.1 THE NEED

The analysis in chapter 2 identified a gap in the tool support for the first stages in the process of designing physical interactions. It provokes the question whether there is a need to fill this gap with an appropriate tool, or whether there is a good reason for this gap.

The case of GUI design

Let's first take a look at the more established area of GUI design. Here, the initial concept phase is often done with the arrangement of Post-It-note on a wall and hand sketches on paper. These tools are very efficient and effective and match very well with the medium that is to be designed (the in-

teractive screen). Even simple interactions can effectively be sketched with the so-called paper prototyping method, where parts of the paper sketch are exchanged in response to the test person's actions (using a Wizard-of-Oz human computer). For more advanced or less conventional interfaces many software tools are available to make quick and fully interactive sketches (e.g., Adobe Flash). Because of the similarity of the medium, the transition between the off-screen and the on-screen is quite fluent.

Nevertheless, even these earliest-stage paper sketches can benefit from a transfer to a software tool, as DENIM shows (chapter 2.5). It alleviates some of the inherent problems of physical paper, while trying to conserve its advantages. Drawings are now digitalized, they can be easily modified, structured, shared, refined – and made interactive. Why then is DENIM not widely used by GUI designers? Mostly because, even though it is fully functioning, it is a research project and as such not completely finished and no longer supported. Its user interface is unconventional and requires learning. Commercial tools are adopting more and more of the early-stage spectrum, as in iRise's scenario functionality, but they are not as radical as DENIM. Another good reason for a slow adoption could be that paper is actually still better than a software tool. Its material properties are nicer, it is calm, focussed, direct, very high resolution and extremely flexible (see also my analysis in Knörig, 2006, ch. 4.2). It cannot be easily shared over distances, but it is superior in supporting collaboration on location[1]. And the cost of digitalization is not so high because the next refinement stage requires the creation of a new document anyway.

Thus, all in all the advantages of using a software tool like DENIM for GUI design are not overly convincing at this point (but would be more so if it was perfected in a commercial product). Paper is sufficiently easy to handle and in some aspects unmatched by software tools, and more importantly, it mimics the design medium very well.

Sketching physical interactions

So how can this realization be transferred to the realm of physical interaction design? The main difference, as laid out in chapter 2.4, is the exponentially growing freedom and complexity. GUIs can be simulated in most

1 There actually exists another research project, the "Designer's Outpost", which takes DENIM to a digital whiteboard with physical Post-It notes (Klemmer et al., 2001).
http://dub.washington.edu:2007/projects/outpost

cases by a succession of drawings on paper, but what about an alarm clock that drives out of reach when you try to snooze it[2]? A pencil sketch can be used to capture the idea, but to make it experienceable the sketch needs to resemble the design medium more. Therefore, designers will often build rough bricolages made of any available material to explore the idea and get a sense of what the object would be like. They also enact scenarios with these bricolages and record them on video, again to learn more about the created experience and to make it discussable with others. It is important that sketching at this stage is quick and fluent.

Current difficulties

These methods are quite effective in evoking the intended experiences and help the designer to go on, but compared to GUI sketches they have some disadvantages. Firstly, they are usually not interactive and the Wizard-of-Oz technique is not practical for testing. Furthermore, unlike drawings they are difficult to document and archive, because they take up a lot of space and need additional written explanation. They are also difficult to rework and refine, because they cannot easily be copied and, depending on the material, modified. Finally, because of their often pointed nature, they tend to divert the attention from a systematic view so that the designer might overlook side aspects.

While a chaotic sketching situation is good and appreciated, these sketches need to be transferred into a concrete and more structured design at some point. This requires the designer to review the created bricolages and videos and refine them in drawings and texts. It is difficult to keep the connection between all these materials and to stay on top of the evolution of the design. The chaos at this point is detrimental to the design process, because important decisions and ideas might get lost, and problems or gaps remain undiscovered. In other words, there is a discrepancy between the messy nature of the plentiful sketches and their synthesis into a coherent concept. This transition requires a thoughtful approach that is not always opportune in the midst of the process.

Opportunity

A new tool should seek to make this transition more conscious and help the designer to transfer his sketches and ideas into a sound design. The tool

2 Clocky. http://www.nandahome.com/products/clocky

should allow the collection and capturing of sketches, where the experiential qualities of the sketches should be retained as far as possible.

It should then accompany a thoughtful continuation of the design process, and provide the ability to compare designs and refine them, to explore variants and alternatives, and to enrich it with context. This would support the design of physical interactions in a similarly concerted way as it is now in GUI design.

4.2 BASIC CONCEPT

The basic idea for the tool, tentatively called Sketchbook, is to provide a place for collecting design artifacts where they can be structured and interactively experienced. It accompanies the established sketching methods and acts as a central hub for the design activities. Both structuring and interactivity start out informally and in a sketchy fashion, and can gradually be formalized. A similar approach is taken by the concept of Dow et al. (2006) for the design of ubiquitous computing applications. This way the tool supports the design along the conceptual phase and well into the sketching phase (Figure 4.1).

Sketches will further be done in all kinds of bricolage materials, because the bodily experience is crucial. But structured thinking is better done on a plane, drawing with a pen on paper or at a computer. This is where we have acquired the most skills and the most control in externalizing our thoughts and reflecting on them. It is therefore that it makes sense to bring the sketches to the plane where they can be arranged and reflected on.

The designer continuously builds a digital sketchbook that contains semi-structured, interactive representations of the design alternatives. Non-digital sketches are digitalized as photographs, scans, or videos, and embedded into the sketchbook alongside with digital material such as drawings and 3d models. The tool then lets the designer assemble simple scenarios and storyboards from this material to explore their validity. He might add variations to the story and enrich it with contextual material. The stories can be tried out interactively by others. Several alternatives are compared and some are rejected. As the design progresses, he refines them to cover more use cases and side activities, gradually formalizing the design.

Through this process the sketchbook evolves into an interactive specification of the design, with rich contextual clues. The complete evolution is

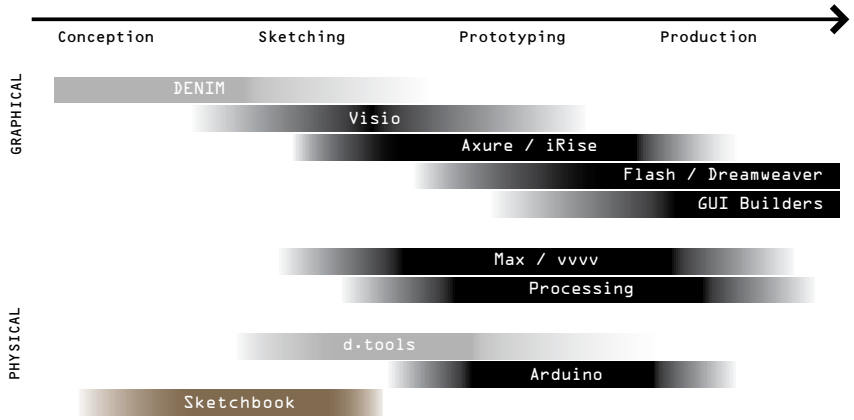


Figure 4.1: Sketchbook fills the initial gap

recorded, with decisions and earlier versions. For the further design process the sketchbook functions as a guide and documentation, also for the collaboration with other stakeholders and disciplines.

4.3 ELEMENTS OF SKETCHBOOK

A walk-through will show several aspects of using the Sketchbook software. To illustrate the use case, I have chosen a mobile phone project of mine, the Dynamic Knobs (Hemmert et al., 2008). It is a phone featuring a new kind of physical interaction: It uses change of shape (a knob) to permanently display a change of state. It is a true tangible user interface in the sense that digital information has a tangible expression that is both output and input. The knob reveals itself when the phone has received, e.g., a text message, and the user can react to it by pressing precisely that knob.

Collection

The standard way to start using a sketchbook would be by collecting assets. These could be any kind of digital material, like photos, videos, PDF documents, links to web sites, 3d models, etc. Physical material needs to be digitalized with photos, videos, or scans. Figure 4.2 is showing a collection of inspirational photographs that were collected during the initial brainstorming phase to set the theme of the design. In this case, they are

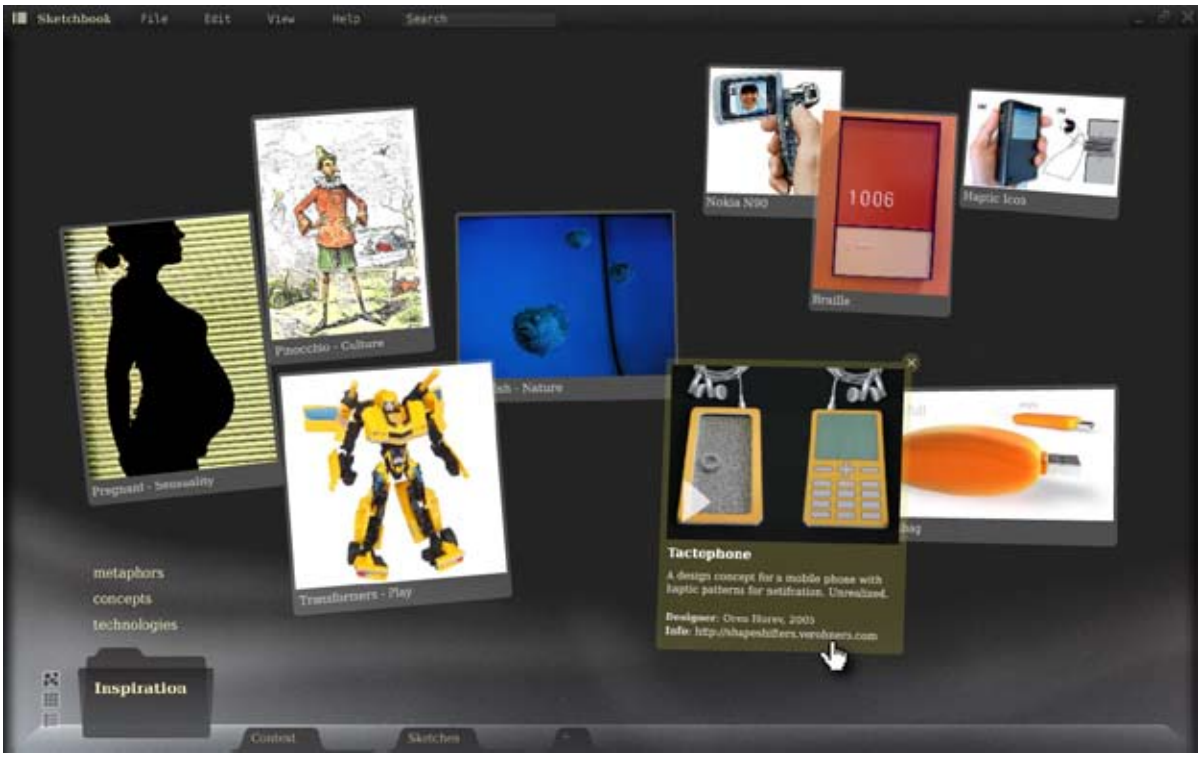


Figure 4.2: Showing a collection of assets

all related to the phenomenon of shape change.

The collected items are stored in folders that are lined up at the bottom of the sketch area. These folders are freely definable and separate the collections into general categories like inspiration, context, or sketches. Within a folder contents can be tagged to create clusters for easier navigation and filtering (in this case we have metaphors, technologies, and concepts). Additional comments can be added to individual items.

The sketch area always has a very tangible feel. Items can be freely arranged, positioned and scaled so support spatial structuring. This way, items can quickly be sorted and related to each other in an informal way, as in the physical world. It relieves the designer from making decisions that are not clear to him yet, and enables the use of spatial memory for orientation and retrieval.

In case the collection outgrows the available screen space, further enhancements of the interaction principles might be borrowed from other

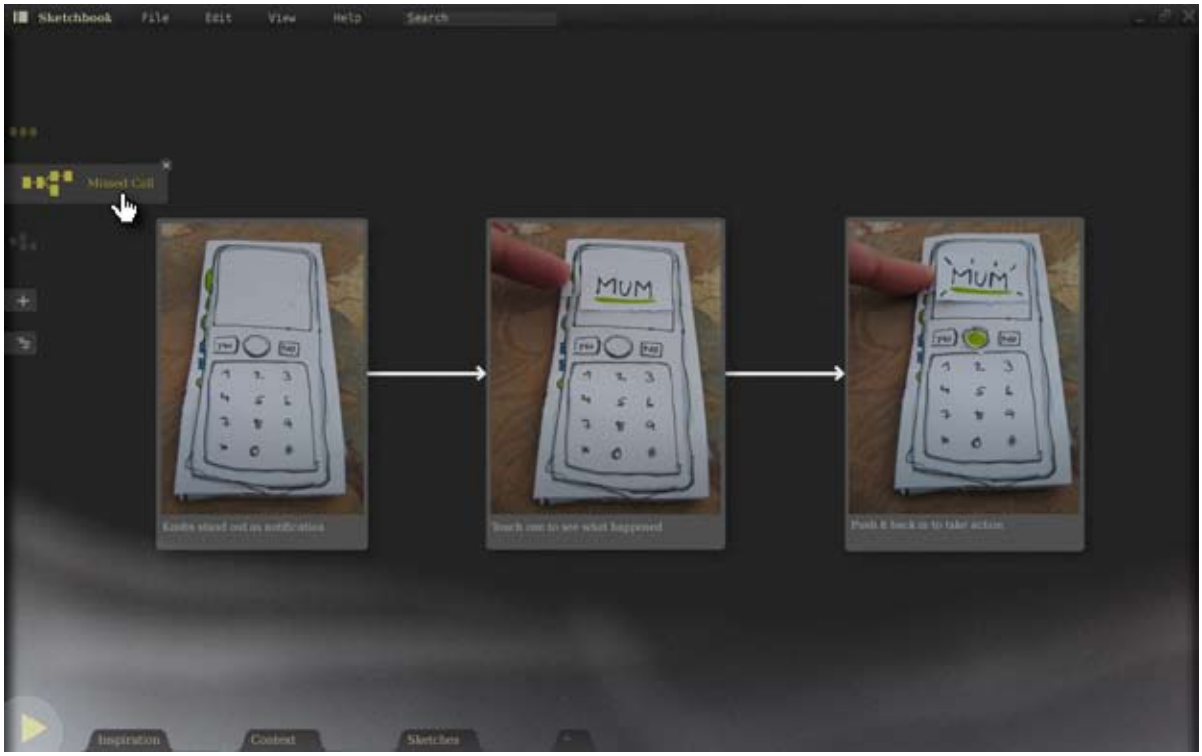


Figure 4.3: Starting with a simple, informal story

projects. The stacking metaphor from the physical desktop simulation BumpTop Desktop[3] (Agarawala & Balakrishnan, 2006), as well as a Zooming UI[4] (Bederson et al., 1996) are feasible techniques.

First sketches

From the collection of material the designer can start assembling the first sketches. These are loosely defined graphs where the items are arranged in a spatial sequence and may be connected with arrows. Comments can be added to explain what is happening. Figure 4.3 shows a simple interaction sequence that was created from a physical paper sketch created before. This simple assembly already presents an advantage to the mere paper version. It allows a quick and comfortable documentation of an idea that would

3 http://www.bumptop.com
4 http://en.wikipedia.org/wiki/Zooming_user_interface

otherwise have to be done in a special illustration. The presentation functionality (shown later) make the sketch experienceable.

Sketchbook also provides prominent support of a very central design behavior that is only insufficiently supported with keeping separate files – variations. They are collected in easy reach on the left side of the screen, for quick switching and creation of new alternatives. Miniature representations of the sketch make them quickly identifiable.

Interaction notation

Building on the first simple interaction sketches, the designer can now apply additional graphical notation to further define the interactions. The created diagrams are essentially state-transition-diagrams: The blocks define the different states that the object can have or that the world is in, the arrows define the transition from one state to another.

Through adding syntax, the diagram becomes semantically richer. The transitions are what interaction design is mostly about and therefore need to be defined in more detail. Transitions are events that lead to a change of state. They can be caused by the user, the object itself (e.g., a timed alarm), or other external parties (e.g., the network). Another way to look at it would be an input-output-cycle, but this view is too rigid. It could be argued that also the state-transition-diagram is too narrow, especially with reference to Merleau-Ponty's theories of experience (1945) which shows that sensing and doing cannot cannot be separated. The notation here provides a practical way to design interactions in an easy way. As will be shown, the language is flexible enough to describe more complex interactions. It must also be remembered that the sketchbook is used as a complementary tool to physical sketches, whose experiential qualities cannot be captured in a visual representation.

Figure 4.4 shows the first step of how the example is transformed into a more detailed description. The visual language has to answer several questions. What is happening and how? This is shown in the imagery and described in the text box. Who causes the change (actor)? This is denoted by the shape of the event, specifically the notch in the event circle. Finally, when does it happen (time)? This is answered by the shape of the arrow. The break before the event denotes that nothing will happen automatically if the event is not actively effected, and the straight line afterwards means that the next state is reached immediately after the event. The fact that these separate concerns of what/how–who–when are graphically repre-

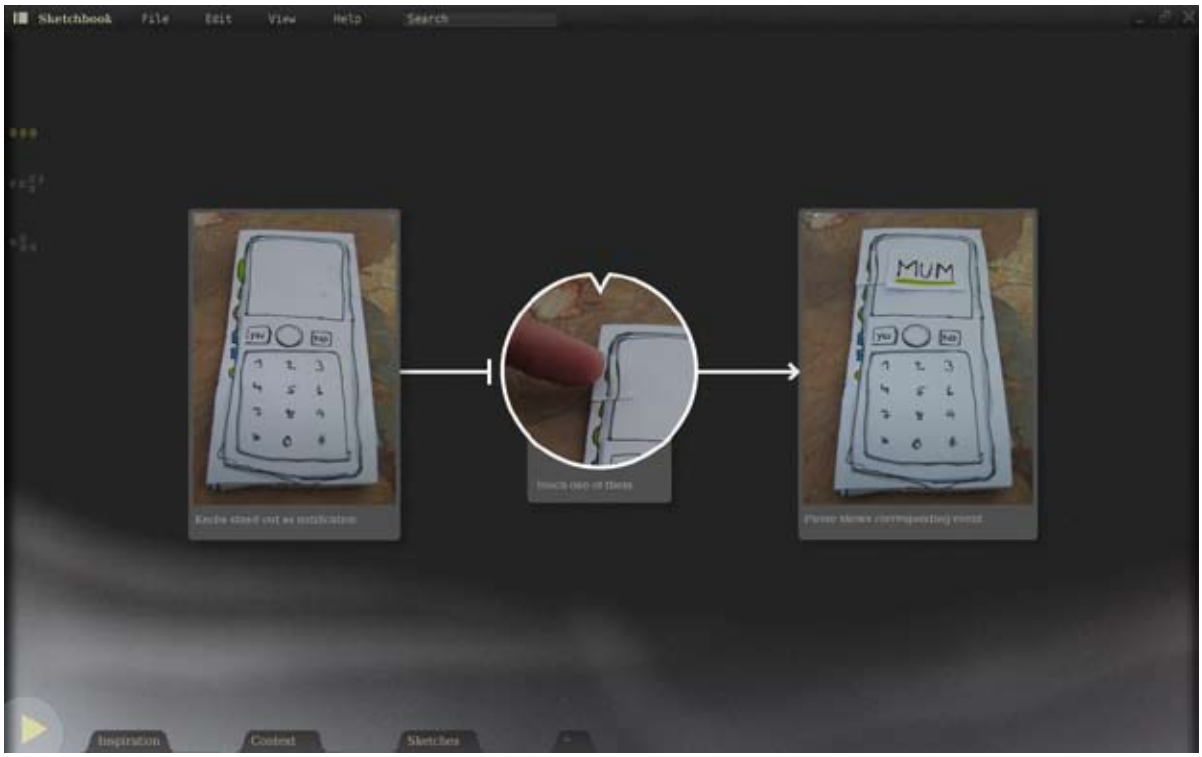


Figure 4.4: Adding a simple interaction

sented in different ways gives room for further variation within each.

Figure 4.5 shows a more complex diagram that goes into more detail on some of the interactions: The phone is initially in the pocket – a network event reaches it while the owner is playing soccer – the knob stands out – it is felt through the pocket – the phone is taken out and the message is previewed – the user is taking action. Through continuous refinement like this the designer can evolve stories without the need to program or formalize anything, yet with a meaningful structure.

The resulting diagram slightly looks like a comic strip. This is intentional, because they are very powerful in conveying a narrative on a small area. Further comic design principles like balloons or overlaid panels could be borrowed to enhance the diagram , e.g., as in the comic creation software Comic Life[5]. There is also related the notion of "design comics" which refers

5 http://plasq.com/comiclife

Figure 4.5: A more detailed interaction scenario

to using comics to illustrate design scenarios.[6]

As the diagrams grow more complex, it could be helpful to borrow the level-of-detail-zoom from DENIM. The diagram could be separated into modules that appear when zoomed out, and explode into the individual elements when zoomed in.

Context

Unlike GUIs, the situation in which a physical interface product is used is very dynamic. The user carries it around and might use it on-the-go in different places, not just at the desk. This requires the designer to explore the possible contexts and integrate them into the scenarios. In his sketchbook, he can simply paste contextual media into the diagram, and associate people, places, activities, and situations with it. Figure 4.6 shows how

6 http://en.wikipedia.org/wiki/Design_comics

Figure 4.6: Enriching the story with context

the designer brings a context video to the front to remind himself of the scenario.

History

The design decisions that have been taken over the course of using Sketchbook are stored in a transparent version repository. Figure 4.7 shows how the decision history can be browsed after opening it from the stories bar on the left. Every story created for the project is traced here. The yellow vertical line show the current point in time so that it is clear which stories have been rejected along the process. A fish-eye zoom brings the last week into focus and the user has selected one of the stories to look into the changes that have been made for this particular one. He could now bring back rejected stories or compare different versions, much like in a code version repository.

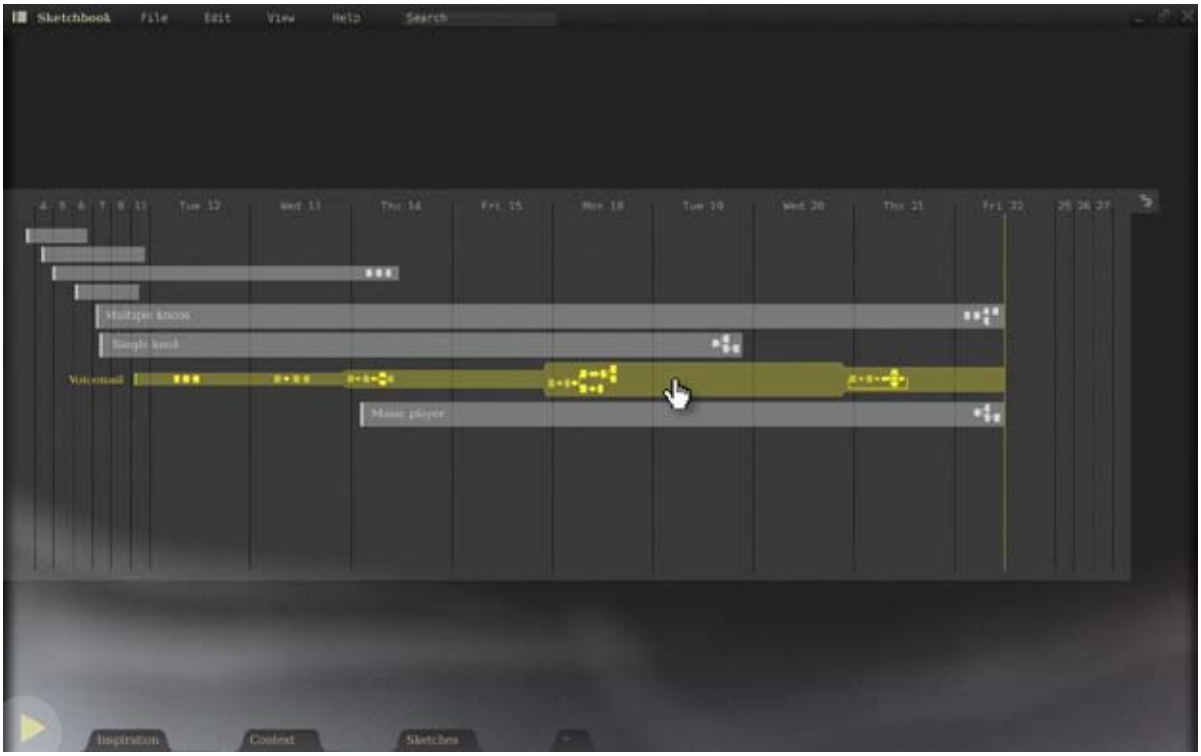


Figure 4.7: Reviewing the design history

Presentation

A story can be presented at any time by switching to the presentation view (Figure 4.8). This allows the designer and others to experience an idea interactively in a click-through-manner. The current step is shown across the full screen, with all distracting elements from the design view hidden. The user can now click through the story by choosing the path to follow. At the bottom the accompanying description is displayed along with a simple web-browser-like navigation. The presentation view also supports quick annotation of the design through little sticky notes.

The presentation is simple, yet effective. It lets the designer easily switch the perspective from fiddling with the design to experiencing it from a user's point of view. This helps him to always be reminded of the experience he is creating and reinforces Schön's seeing-drawing-seeing-cycle.

Figure 4.8: Presentation view with annotation

Logic

At this point the designer might want to move on to further specify the design in terms of programming logic. The mere diagram is good for initial storyboarding, but will eventually grow beyond maintainability. As the design process is nearing the prototyping stage, it would be helpful to turn the sketches into a more programming-like notation.

Sketchbook supports this through gradually adding programmability. As a first step the user needs to describe the elements of the world in an object-oriented way (Figure 4.9), i.e. a hierarchy of objects with properties and actions that can be performed on them. This becomes more powerful when the corresponding sketch is a vector graphic where individual graphical elements can be assigned to the object tree.

With this description of the world the designer can now go back to the diagram and add programming logic to it. Figure 4.10 shows how state in-

Figure 4.9: Adding object-oriented knowledge about the world

formation is added in the left drawer and the interaction logic in the right drawer. The programming can be done semi-graphically, similar to that of Scratch (see chapter 2.5). The script's basic building blocks follow the diagram metaphor: 'on' defines the event, and 'do' transforms the state into a new one. Further language elements like conditions and loops are available. The programming objects are those that were defined in the object hierarchy and it is possible to create simple animations through demonstration. The scripting language could be further extended into the direction of Scratch to allow more complex animation. However, the idea is to stay state-based. Nevertheless, the programming blocks help to simplify the graph.

The addition of formalized logic buys us two things: Firstly, the presentation becomes more interactive and more meaningful. Figure 4.11 shows how the presentation is enhanced. The user's action can now contain directly interactive parts instead of just navigating through the diagram. It

Figure 4.10: State-based programming of the interaction

is also clearer what the action actually does and how it transforms the objects' states.

Secondly, the way is paved to move on directly with prototyping. The code that is created here could be exported to different prototyping platforms. The ideal partner would be d.tools (see chapter 2.5), because it follows a similar programming metaphor. The states and transitions could be conserved and the designer would only need to match sketchbook objects with the physical widgets available in the d.tools toolkit, and could immediately build a fully functioning electronic prototype.

Figure 4.11: Enhanced interactive presentation

TOOL II: FRITZING

5

This chapter presents a study of the practical design considerations for a tool that I have co-developed from inception to the release of several versions[1]. As a balance to the previous chapter, it discusses a complete real-world use case. The design of the tool was therefore not so much guided by the vision to create a revolutionary user interface design, but rather to create a fully functioning, practical tool for designers – with the given resources. These constraints require pragmatic decisions and a constant evaluation of what is the most effective, rather than what is the most interesting. As a result, Fritzing has become a highly useful tool.

5.1 THE NEED

The idea for Fritzing resulted from a dissatisfaction with the fidelity of current physical interaction design prototypes. In recent years, more and more designers have experimented with physical interactions. Before, it was often reserved to HCI researchers who had enough technological understanding. With the widespread availability of simple micro-controller

1 DISCLAIMER: The Fritzing project is a research project at the Interaction Design Lab of FH Potsdam, funded by the Brandenburg Ministry of Science, Research and Culture. Between my Master studies I worked on Fritzing as the project lead, under supervision of Prof. Reto Wettach. I was responsible for the overall concept, the interaction design, the development and the web site. This chapter is a critical discussion of the work carried out in that time, in the light of the presented tool design guidelines.

Figure 5.2: A typical breadboard-based prototype

platforms such as BasicStamp, Arduino, and Wiring, and the ideas and concepts developed at schools like the Interaction Design Institute Ivrea, physical interaction design could establish itself quickly as a design discipline.

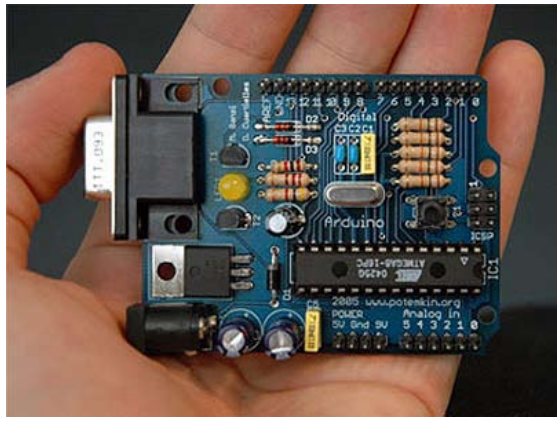


Figure 5.1: Arduino micro-controller
Photo by Nicholas Zambetti

Current situation

The Arduino platform (Figure 5.1) proved to be the most successful tool in this regard, because it struck the best balance of simplicity, power, cost and community (Mellis et al., 2007). From its release around 2005, it quickly became the standard tool among designers, but also artists and electronics hobbyists, with far more than 10,000 sold and uncountable custom-made ones. The main contribution of this tool was that it for the first time enabled a broad base of designers to work with electronics, to use it as a material almost like wood and metal. Hardly any knowledge of electronics need to be learned to make a light blink, sense the physical world through a range of sensors, and drive motors.

Thus, designers can nowadays rather easily build prototypes of their physical interaction ideas, and depending on their skills these can become astonishingly advanced. The limitation with the current state is the level of fidelity that these prototypes can achieve. A prototype is usually built in the following manner: A professionally produced Arduino functions as the heart and contains the logic. Via wires it is connected to a breadboard, which is a simple standardized plastic board with interconnected plugs. This breadboard contains the circuit, an array of electronic parts and wires plugged into it manually (Figure 5.2). The advantage of the breadboard is that it perfectly suits the working style of designers: Simply try and re-try and continuously refine until satisfaction – sketching in hardware.

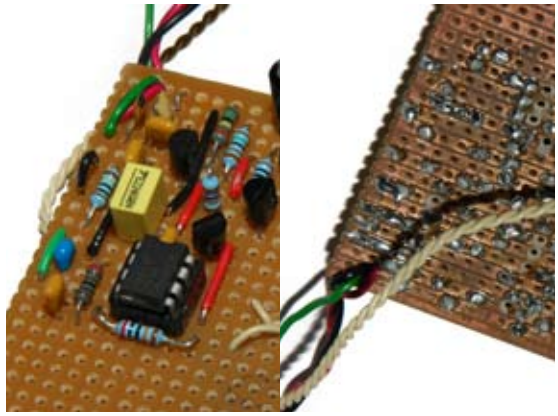


Figure 5.3: Stripboard (front & back)
Wikimedia Commons

Current problems

The disadvantage is obvious: The prototypes look messy, they are big and clunky, and easily fall apart. Presenting them to customers or in an exhibition is

Figure 5.4: PCB panel

risky and often results in problems. Also, the charm of looking like it's fresh from the lab only goes so far. This can be compensated partly by the use of stripboards, but they are much more cumbersome to use (Figure 5.3). Another problem is documentation: Taking photographs or making hand-drawings are poor solutions and make sharing with peers difficult. And even finished circuits are often taken apart to reuse components.

In the short time of a few years, designers have already reached limits in their work with electronics. If they want to move beyond, they currently need to hire an engineer or learn using professional engineering tools themselves. As this requires a thorough foundation in engineering, this is reserved to few and also diverts their time from the design work.

Opportunity

Essentially, designers are missing support in what electrical engineers are calling electronic design automation (EDA). This category of tools lets the engineer design an electronic circuit and then generate the data necessary to produce printed circuit boards (PCBs) from it (Figure 5.4). This is the world-wide professional standard for designing, documenting, and producing electronics.

Therefore, the next logical step in supporting physical interaction designers is to give them an EDA tool – with a designerly twist. Such a tool can fill the gap identified in chapter 2 (Figure 5.5), as an electronic analogy to what Adobe Dreamweaver is for designing web pages.

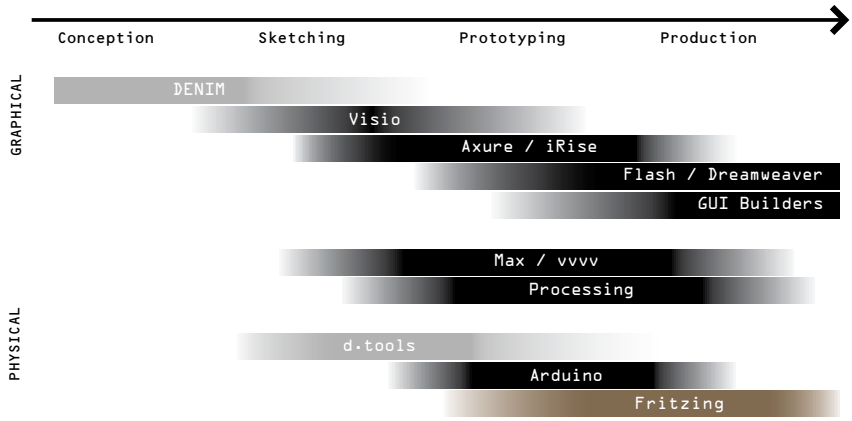


Figure 5.5: Fritzing fills part of the production gap

5.2 THE FOCUS

With the given problem and a general idea for a tool, the question is what kind of tool it will be like for the designer, and which core competence it should provide. We start out by exploring the implications that an EDA tool could have for the current situation, and later bring them into perspective of what is the most important functionality and how others could be situated around it.

Carving out the territory

As described, the original motivation was to move the designer closer to a position of a producer, i.e., to enable him to create higher-fidelity artifacts. The standard for this is the PCB, so the first requirement is to output the necessary data to have the PCB produced externally or in the in-house lab. Since the current mode of working with breadboards should be conserved, the tool needs to provide a bridge between breadboard use and PCB layout. This observation resulted in the idea to create a graphical editor that lets the designer replicate his breadboard sketch on the screen, and semi-automatically guide him through the PCB-making process from there.

Once the designer has recreated his circuit with such a graphical editor, it only needs little additional information like a description and part information (the "bill of materials") and the project's electronics would be documented and becomes archivable. There is no longer the need to be overly careful with the original breadboard, as it now can be recreated from the documentation at any time.

If the documentation is done right, the file format defines a standard that can also be shared with others and support collaboration. If combined with the source code that runs on the micro-controller, it would be the complete blueprint for the design. It would pave the way for open-source hardware, and an online community could be built around it specifically targeted at designers. Besides openly sharing designs, the web site could provide a platform for members to share their general knowledge in using electronics for design. Some of this knowledge could even be fed back into the tool itself.

Another use case that follows from this is to support the learning of electronics through providing tutorials and examples. Likewise, the tool could be used for teaching and have additional features for demonstrations on a projector. For example, a schematic view could automatically be generated in parallel.

More advanced features could allow the simulation of circuits on the computer, the support for product design by providing 3d-measurements of the PCB, or the experimentation with other circuit materials such as cloth.

Refocusing

This brief exploration shows how the concept for a tool leads to a wealth of use cases and interesting related features. A tool automatically gives rise to its own little ecosystem that can be actively grown, but in the beginning it is important to focus the resources and determine the main contribution. In the case of Fritzing, this was identified as the easy ability to make PCBs, because it provides the greatest tangible benefit to the designer. The tool helps him to achieve something necessary that he could not, or only with great effort, achieve otherwise. The core idea of taking the user from his breadboard prototype to professional PCB production was then illustrated as a memorable comic strip (Figure 5.6).

The other possible features are chosen by comparing benefit with cost. Documentation basically comes for free, and the careful design of the file format provides numerous other benefits such as compatibility and scriptability. Greater costs are involved with a simple sharing functionality, as it means finding a way to deal with the endless variations and versions of electronic parts. However, easy sharing is the foundation of many other benefits and is therefore a feature that needs to be integrated from the start.

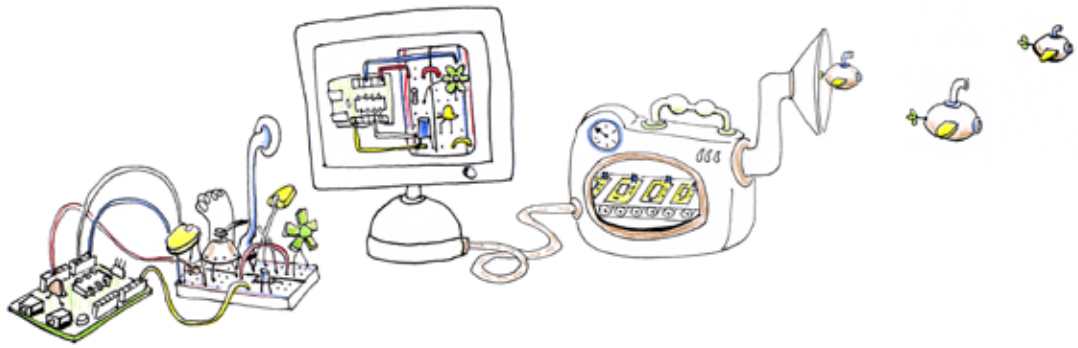


Figure 5.6: How Fritzing works
Illustration by Myriel Milicevic

Building an online community is central to making the tool known, but it is also an important way to learn. Especially when it comes to the use of technology, designers prefer to learn from how others have done something, rather than reading a book on the topic. They can benefit greatly from a community that shares their examples.

The other features can be developed over time, and can partly be grown by the online community. Learning and teaching are side-topics that can be kept in mind, and advanced topics might be attended to later as the need or the opportunity arises. Simulation, for example, might seem very powerful at first sight, but would require an enormous effort. And because designers need to get hands-one with the breadboard anyway, its benefit is questionable.

Fitting into the landscape

A new tool is usually not defining its own new context, instead it has to blend into an existing environment of other tools and processes. The situation for Fritzing was already briefly described: It sits at the edge between the use of micro-controllers in design and the use of EDA software in engineering. For the former, Arduino provides an ideal opportunity to build on. It is a mature tool that is respected and well-established among physical interaction designers. Also, it is easy to learn and by its design lends itself well to an integration with Fritzing: The concept of the "Arduino shield", a standard for an extension PCB that fits on top of the Arduino (Figure 5.7), can be used as a simple default for beginners. Arduino thus is the perfect starting point, but Fritzing should not become too tightly integrated, and stay open for other uses. Additionally, Arduino is a great role model for a

Figure 5.7: Arduino prototyping shield
Photo by Limor Fried (ladyada)

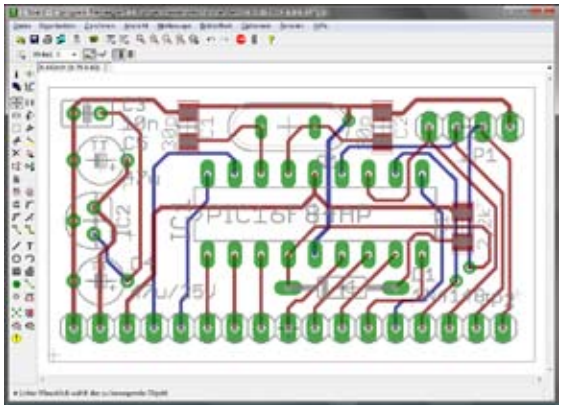


Figure 5.8: EAGLE (showing a PCB view)
Cadsoft

successful tool ecosystem. It itself followed the model of Processing, and Fritzing will also benefit from joining the team due to the community that already exists.

At the other end, a research into the market of existing EDA software showed a very wide range of tools, with high-end commercial products, freeware provided by and for hobbyists, learning kits targeted at schools, etc. Very interesting proved to be the freeware and open-source products, for two reasons: Firstly, designers would not pay a high license fee to use a product that is only at the border of their needs, and secondly, these tools could be employed by Fritzing to do the heavy lifting of the PCB generation process. The Fritzing team could then focus on its strength, the user interface design.

The most popular EDA tool among technically advanced designers is EAGLE (Figure 5.8), mostly because it is relatively cheap and has a freeware license for limited use. Even though it is not open-source, it can be interfaced with through a relatively simple and powerful scripting language. We therefore decided to start with building our efforts on EAGLE, and later move towards a closer integration with one of the open-source tools.

5.3 THE DESIGN

Now that the focus and the environment of the tool have been defined, the design and development of the tool can be approached. Fritzing has

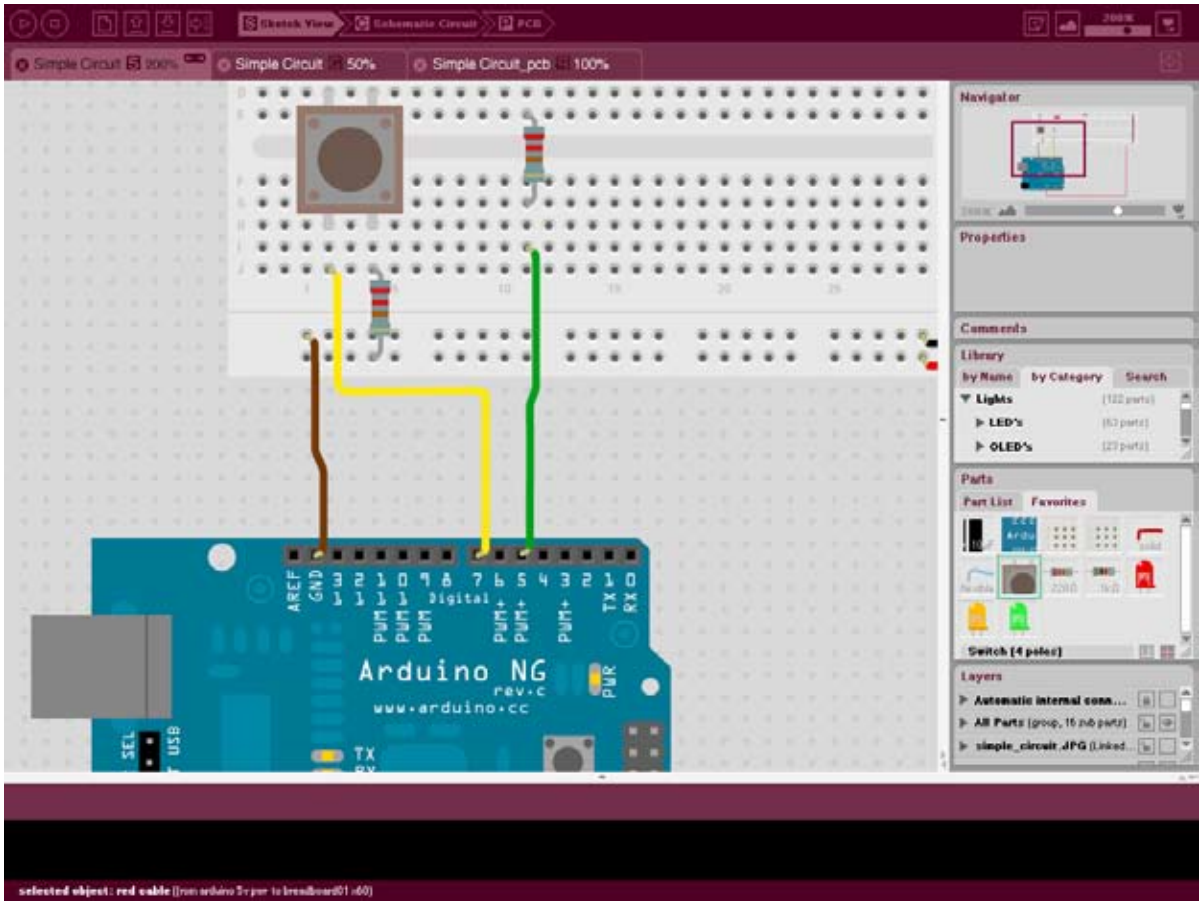


Figure 5.9: Sketch of the Fritzing GUI
By Dirk van Oosterbosch

been developed under practical constraints and budget restrictions, and is still at an early stage. Not all of the following aspects derived from the guidelines in chapter 3 are therefore present in the latest available version.

Low threshold

The main challenge of the tool is to make complex technology usable by non-technologists. Fritzing was therefore designed to integrate seamlessly with the current practice and process and pick up the designer right where he is left alone. It is a top priority that the tool can be used by anybody

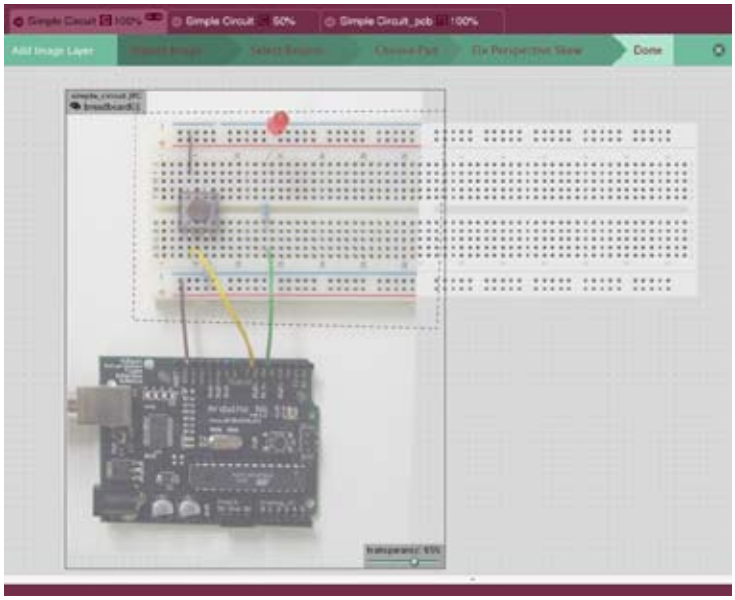


Figure 5.10: Using a photo as blueprint
By Dirk van Oosterbosch

who just knows how to make an LED blink with Arduino. This is achieved by a graphical editor that resembles the real world situation in look and feel (Figure 5.9). Parts that look like their real counterparts can be dragged from a simple parts palette onto a large sketch area. They can be rearranged and wires can be drawn among them, until the virtual sketch is identical with the physical one. This "breadboard view" does not exist in professional EDA packages, but it is sufficient for the electronics amateur, providing a simple, safe, and playful environment.

An idea that takes this even further is the blueprint function. Users could take a top-view photograph of their physical sketch and display it inside Fritzing as a semi-transparent overlay. This would ease its graphical recreation even more (Figure 5.10).

At this stage the documentation would be completed. If the designer intends to turn it into a PCB, he could switch to the "PCB view" where all elements are already placed. He can then choose a template for the PCB (e.g., the Arduino shield), rearrange the elements, and the software would automatically take care of the rest. Even though there are endless options that could be tweaked for this step, we learned that even engineers usually

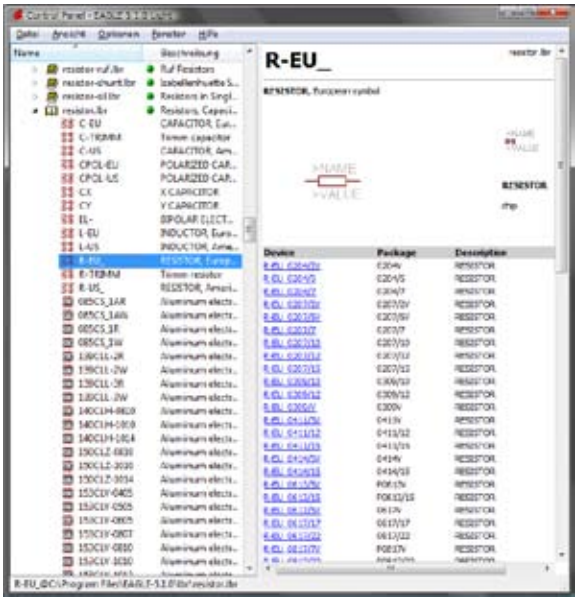


Figure 5.11: Browsing for the right part in EAGLE
Cadsoft

resort to the values that they know will work, so we simply provide some sensible defaults.

A more advanced concept for lowering the threshold is that of modules, pre-composed functional entities for frequently used functionalities such as Bluetooth communication or motion sensing. Modules would essentially be their own Fritzing file, so that a project could be recursively built from modules and sub-modules. For beginners, it would mean even less technical knowledge necessary to explore electronics in their designs.

High ceiling

The described procedure already covers a large percentage of use cases. If the designer wants to move on to more advanced uses, there are several ways inside and outside of Fritzing. A simple way are alternative templates, for instance, one that embeds the whole Arduino circuit, or ones for other Arduino versions like the Arduino Mini. If the templates are not sufficient, a vector graphic can be imported to provide the shape of the PCB.

Fritzing also tries to go a new way for the part library. In current EDA tools these are highly complex and make it very difficult for beginners to

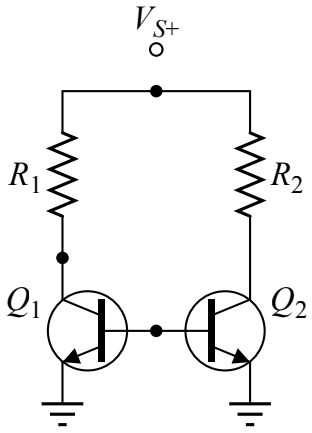


Figure 5.12: Schematic notation of a circuit
Wikimedia Commons

pick the right part (Figure 5.11). Fritzing defines its own simple standard and makes use of part families. There is only one resistor in the library rather than hundreds of specialised ones, in the configuration that is used 90% of the time. If the user needs a more specific one, he can change this in the properties of the part. And if his specific choice is not available, he can use the part editor to make his own. The part editor simply takes graphics files created with other applications, and lets the user define some metadata and the position of connectors.

If the functionality provided in Fritzing is not sufficient, the created designs can easily be exported to formats used in professional tools. As Fritzing makes use of one of these tools for the backend processing, the integration with this tool is quite close. The first difficult steps are then already taken, and the designer can adjust individual details. Such a transition from a simplified tool to a full-blown professional one is usually very steep. In order to ease it, Fritzing will contain a third "schematics view", that lets the user gradually get familiar with the professionals' notation system for circuits (Figure 5.12).

Wide walls

The call for wide walls is answered by loosening the rules of how a circuit can be assembled. Traditional EDA tools are rather strict and only allow compositions that are 'correct' in an engineering sense. Designers however often do not adhere to these rules, be it because of a lack of knowledge or creative experimentation. Fritzing supports these alternative approaches and even suggests them. For example, it allows the user to wire up elements without constraints, so that the leg of an LED can be bent awkwardly and directly connects to a loose wire.

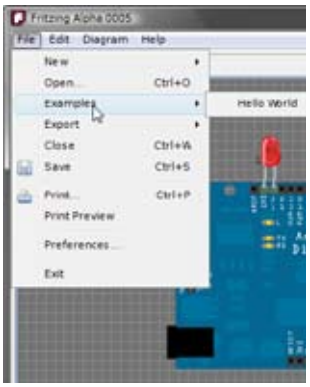


Figure 5.14: Integrated examples
Fritzing Alpha

Another important freedom is given in the choice of composition parts. A designer's circuit may contain 'hacked' and repurposed electronics like a toy puppet or a Nintendo Wii controller, for which a traditional EDA has no representation. For the designer, however, it is an essential part of the project that needs to be documented. In Fritzing, these elements can therefore easily be created with the part editor. Simply take a photograph or draw the element and define its connectors, and it can be used in the sketch. (**)

Exploration and experimentation

These aspects are not quite as important in Fritzing, because it is a tool that is rather used towards the end of the design process. Also, exploration

of electronics has to happen in the physical world. This can be furthered by Fritzing by functioning as a learning and sharing platform for the community. To get it started, the software will contain numerous example circuits that showcase very simple setups as well as advanced circuits (Figure 5.14). This gives users a safe way to learn, to discover what Fritzing has to offer, and to build their own designs on authoritative work.

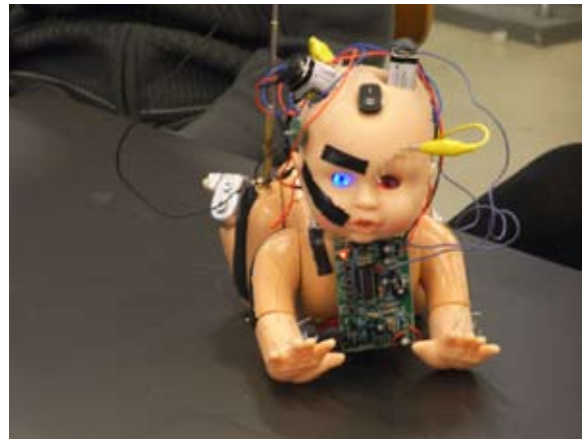


Figure 5.14: Toy hacks can be documented in Fritzing
Photo by Danja Vasiliev

Further support for exploration is instilled by the abstractness of the part library. Users are not forced to search and select a specific resistor, they can just pick the one resistor and become more specific later. This principle could be taken to an extreme where the user can be as unspecific as picking a "light" element, and only later decide if it is an LED or other type of light source.

Trustworthiness is given in part because Fritzing is publicly funded and open-source. Within the tool, it supports standard mechanisms like an infinite undo stack, quick saving, and a project folder that can be archived. A further aspect of trust is the tangibility of the graphical editor: The realistic graphics and behavior give confidence and foreseeability.

Informality

The need for informality is also not as strong as in an early-stage tool. Nevertheless, Fritzing supports it by the already mentioned quasi-realistic graphical editor. It lets the user place elements freely on the infinitely large sketch area, without requiring them to be connected to anything. Parts can also be abstract, even representing hacked objects or conceptual entities. The user is quite free in his composition, unrestricted by formal requirements.

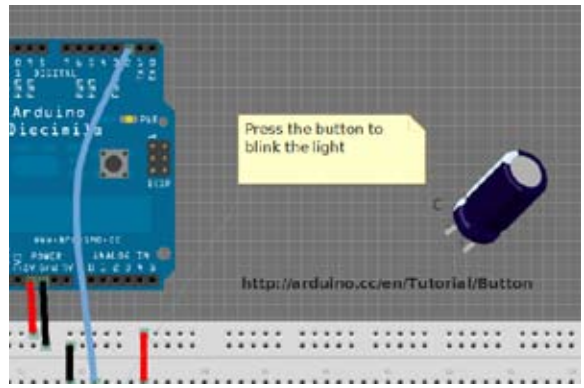


Figure 5.15: Informal annotation
Fritzing Alpha

Furthermore, a note item lets him freely place text notes on the sketch area and visually link them to an element. This allows an informal documentation for alerting others or reminding oneself of something that can not be captured otherwise (Figure 5.15).

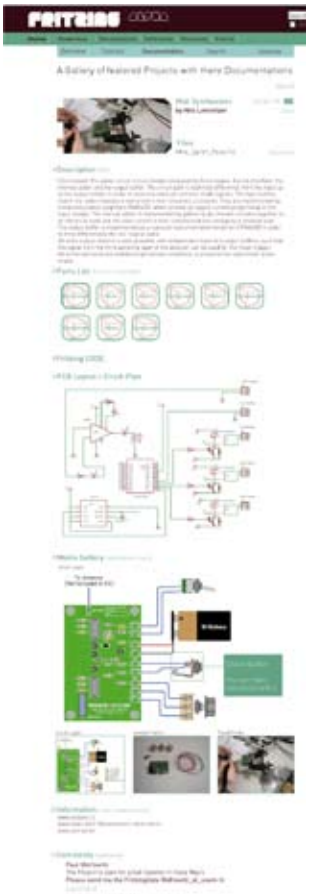


Figure 5.16: Sketch of an online project documentation
By Hendrik Gäbler

Collaboration and community

The sharing of electronics-based interaction designs is one of the key benefits of Fritzing. As already described, this has not been easily possible for designers before and Fritzing makes this for the first time efficient and complete. Designers can start sharing with their peers to learn from another, they can use it for asking help if they have technical problems, and they can collaborate much easier with engineers.

Easy shareability is mainly made possible through a carefully designed file format. Structural information and metadata are stored in a human-readable XML format, and all graphics are stored in the SVG standard. A project folder contains all the information that constitutes the project, including every part description.

A tight integration with the web site will further enhance sharing. It will be possible to upload a design with one click from within the tool, and become part of an online gallery of fully documented projects (Figure 5.16). Similarly, parts can be shared in an online library, and one could even think of an RSS feed for new parts displayed inside of Fritzing.

Detailed control

Because the PCB is usually not visible to a product's customer, it is not a priority to provide easy control on its looks. If this is wished, the user can design the shape in a vector graphics program. The same holds for the graphical aspects of the PCB like the routing of traces and the silk print. The cost of implementing it inside Fritzing would be too high, especially since it would have to compete with professional graphics software that designers are used to work with. This method is sufficiently easy and powerful.

Tool appropriation

The room for appropriation is mainly opened by the possibility to add custom parts to the library. As this is a central part of an EDA, the tool can quickly develop a personal feel for its user.

Since it Fritzing is open-source, users can theoretically customise it as much as they want, but this is very unlikely in a community that lacks the necessary skills. We will therefore explore the need for customized hotkeys, macros, and a scripting language, when there will actually be advanced users. These features can relatively easily be added when the need arises.

Aesthetics

For the graphical design we chose a look that is neither too technical nor too playful. The quasi-realistic graphics are simplistic yet detailed and come in a desaturated color palette. They transport the claim of a professional tool for users who value aesthetics and design (Figure 5.17).

The core library of parts sets a high standard that is defined in a style guide. Users are advised to follow this guide if they want to make their custom parts public. The graphics are high-resolution vector drawings, even with translucent parts, so that sketches look good even when printed.

Fritzing also has its own identity, expressed in a logo, color and font. They were chosen to harmonize with the family of Processing and Arduino (Figure 5.18).

Figure 5.18: Fritzing icon and logo
By Dirk v. Oosterbosch

Flexibility

The tool leaves flexibility to the user by making use of open standards like XML and SVG for the data it creates. Furthermore, it provides open interfaces for both import and export. Typical graphics formats can be imported and exported to ensure high standards in visual design, and on a technical level Fritzing will support import and export of other tools' part definition formats and also schematics and PCB descriptions. This openness allows

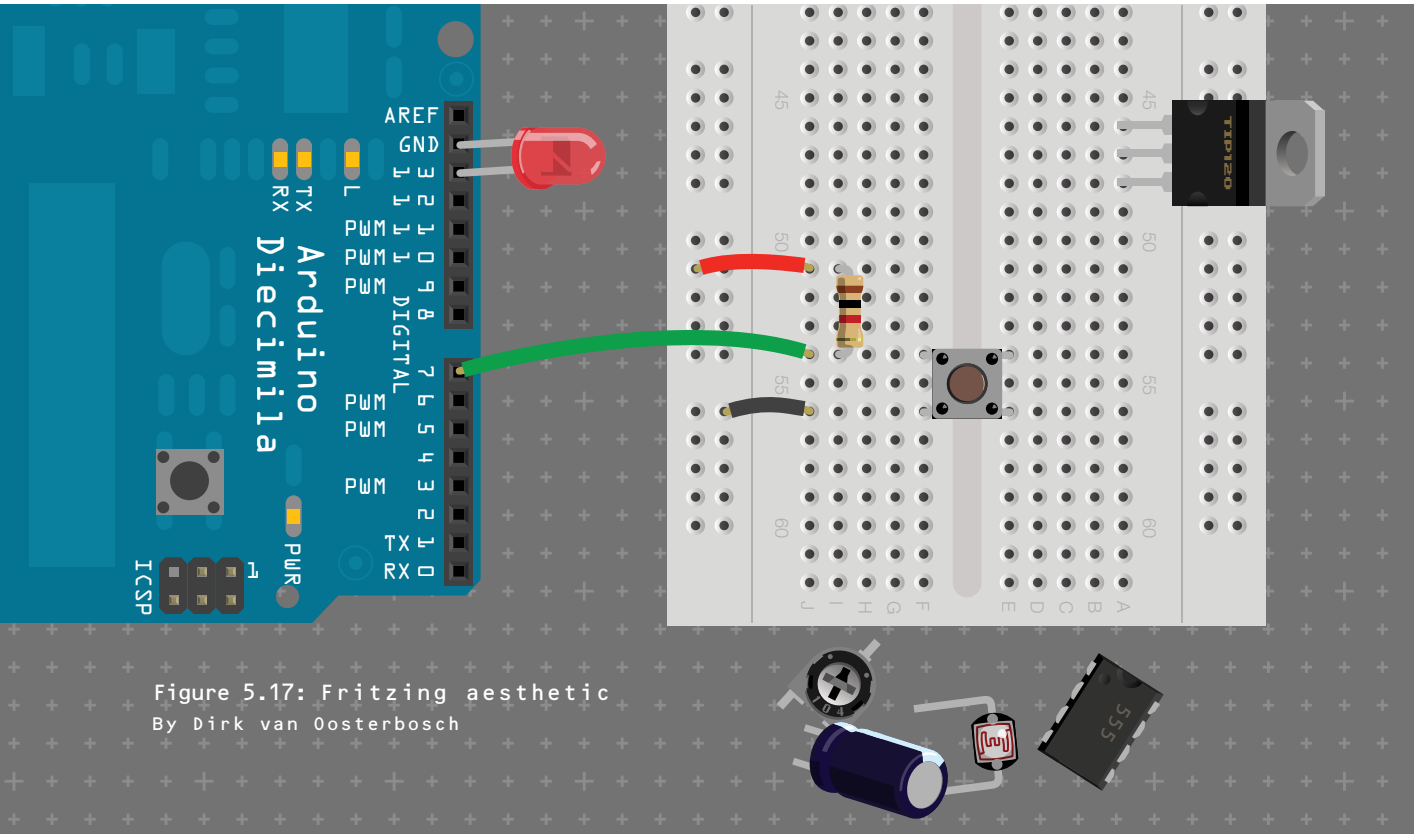


Figure 5.17: Fritzing aesthetic
By Dirk van Oosterbosch

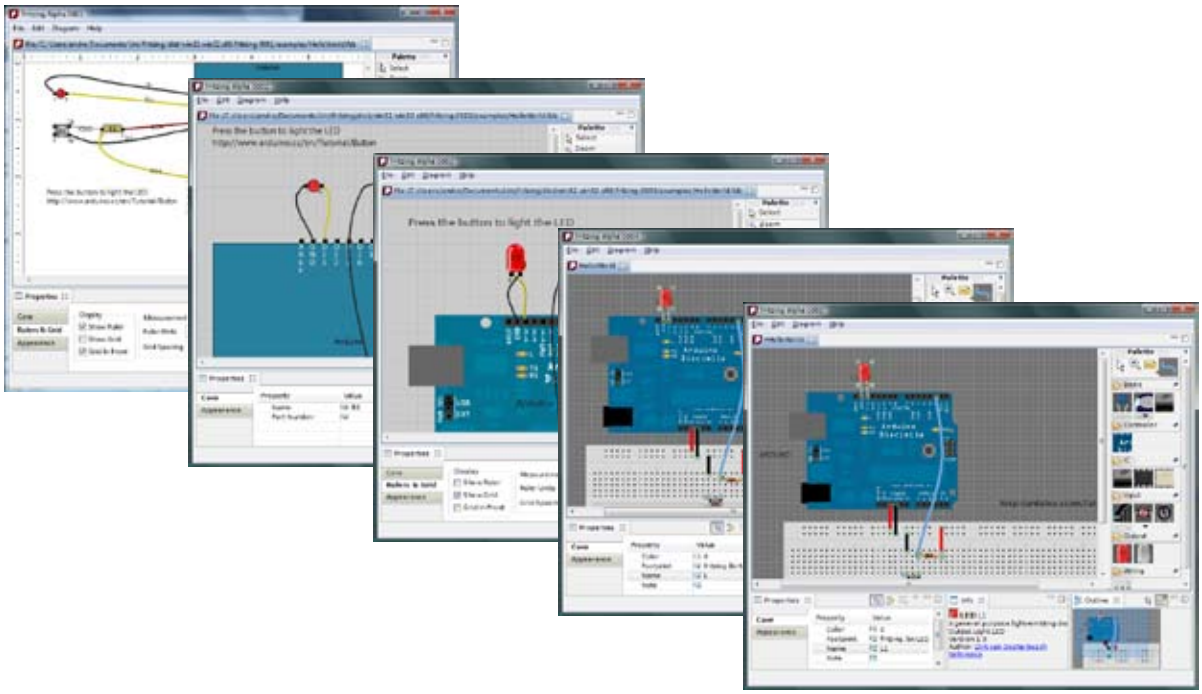


Figure 5.19: The first five Fritzing releases

the user to switch to another tool at any time if that tool is better or more convenient for the job.

(The topics of focus, cost, and periphery are not treated separately here, because they were covered within the other aspects.)

5.4 DEVELOPMENT HISTORY

Agile development

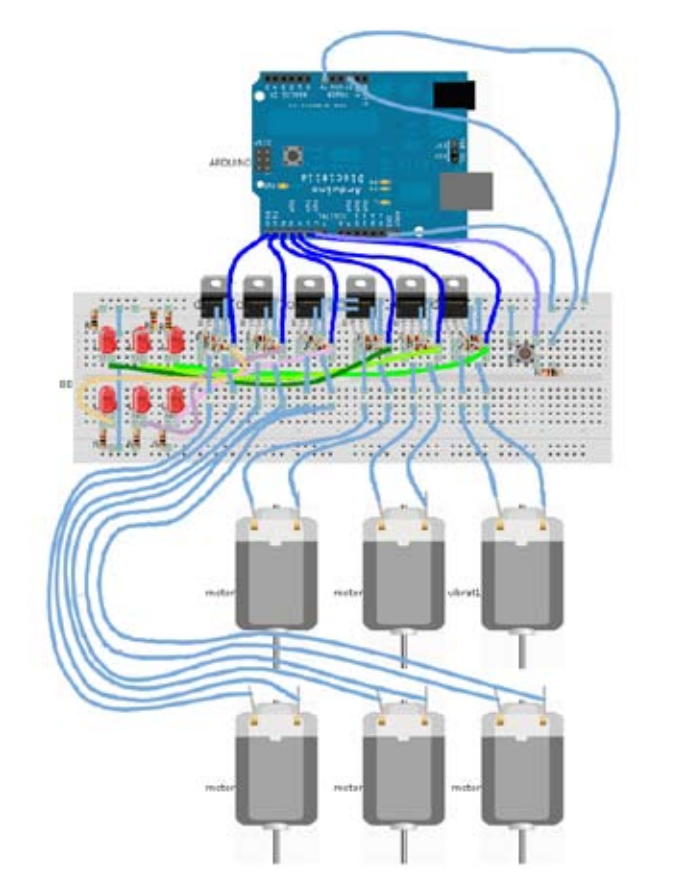
When the focus and the basic concept were clear, the team started developing the tool and the web site was set up. As it was not clear for how long the project would be funded, we decided on adopting a loose agile development approach with short iteration cycles. The first fully functional alpha release was made available only a few weeks after the development started.

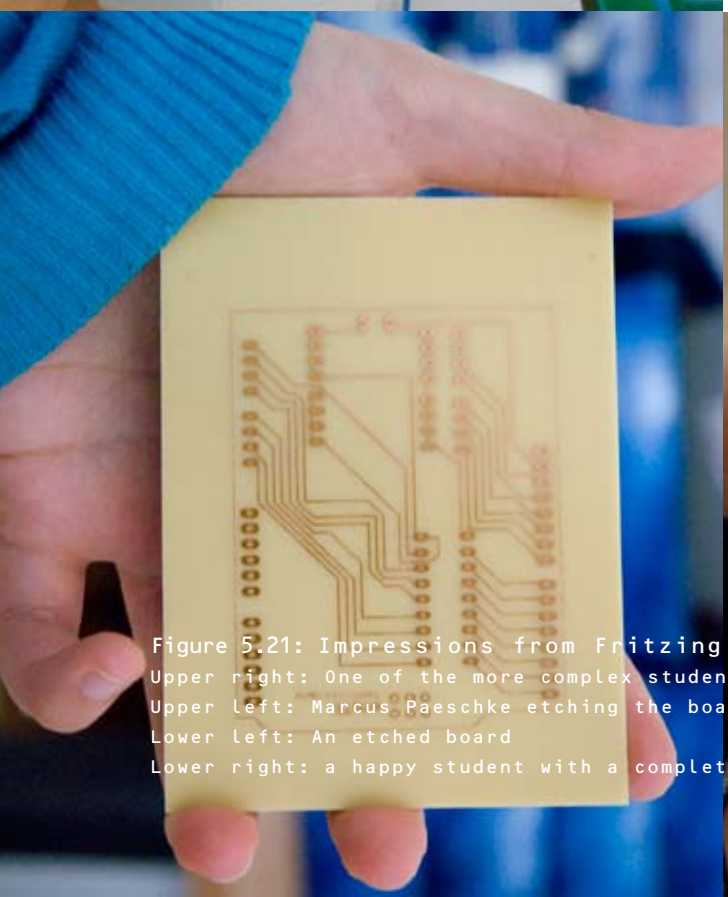
Figure 5.20: Fritzing kick-off workshop with experts
Arduino co-inventor Tom Igoe speaking (Photo by Jochen Fuchs)

(This was possible due to the powerful framework that Fritzing was built on.) After that, a new release was published every three to four weeks, with release number five at the end of the first funding round. This development method proved to be very fruitful: It brought a lot of motivation to the team, focussed the development on practical results, and allowed early feedback from users. After every iteration the focus could be readjusted and refined, which made it an effective design process. (Figure 5.19)

Experts participation

Before the development started, and again before the second round, we held a kick-off-workshop with experts in physical interaction design from Europe and the U.S. About twenty people with backgrounds in design, arts, education, and technology were invited to work with the Fritzing team and local students for two days. The program was constructed so that the external experts could present their own perspective on the subject, followed by intense group brainstormings on Fritzing-related topics. At the end the





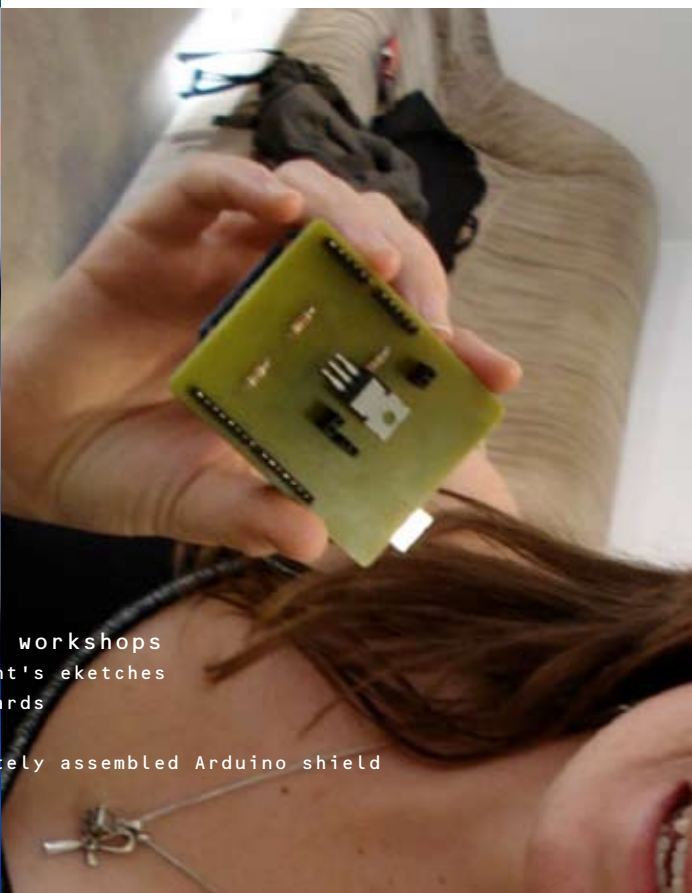


Figure 5.21: Impressions from Fritzing workshops
Upper right: One of the more complex student's eketches
Upper left: Marcus Paeschke etching the boards
Lower left: An etched board
Lower right: a happy student with a completely assembled Arduino shield

results were presented and eagerly discussed. These workshops gave a good indication of where the real problems and opportunities are, and produced a lot of practical ideas. The optimistic spirit and high interest of the participants also provided another motivation for the team. (Figure 5.20)

Continuous testing

The environment of the university is an excellent opportunity for continuously testing the current state with students. Fritzing was given to an undergraduate class in physical interaction design by Prof. Reto Wettach, where they were asked to produce PCBs of their projects by the end of the semester. With some assistance from the team, they were the guinea pigs to try out very early versions of the software. Even though it was at times painful to use, the overall process worked quite well. We gathered a lot of helpful observations and feedback during the study in areas that were not obvious. For example, understandability of the scope of the tool was a key concern.

Workshops

With a later version we gave a workshop at an external place, in the physical computing class of Jan Sieber at the Bauhaus-University in Weimar. This workshop was meant to evaluate if designers can go from minimal Arduino knowledge to a self-produced PCB in two days. Fifteen students were assisted by three tutors (one for electronics help, one for Fritzing, and one for PCB making), and everybody with a concept had his own custom-made Arduino shield at the end of the workshop. The self-production of the PCB proved to be highly motivational for the students.

We will further expand the hosting of workshops because it eases acceptance with users who are otherwise uncomfortable or do not have all the necessary equipment available to them. In turn, it gives us the chance to learn first-hand from the users' experiences. Every user is different in his knowledge and requirements, so that there is always something new to learn on how to make the tool work better for its users.

BIBLIOGRAPHY

Abercrombie, Stanley & Glaser, Milton (1997).
Work, Life, Tools: The things we use to do the things we do. The Monacelli Press, Inc.

Agarawala, Anand & Balakrishnan, Ravin (2006). *Keepin' it Real: Pushing the Desktop Metaphor with Physics, Piles and the Pen*. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '06. ACM, New York, NY, p. 1283-1292. http://www.bumptop.com

Alvarado, Christine (2000).
A natural sketching environment: Bringing the computer into early stages of mechanical design. Master's thesis, Massachusetts Institute of Technology.

Bederson, Benjamin B., Hollan, James D., Perlin, Ken, Meyer, Jonathan, Bacon, David & Furnas, George W. (1996). *Pad++: A Zoomable Graphical Sketchpad for Exploring Alternate Interface Physics*. Journal of Visual Languages and Computing, 7, pp. 3-31.

Boden, Margaret (2004).
The creative mind: myths and mechanisms. 2nd edition. London: Routledge.

Brooks, Jr., Frederick P. (1977).
The computer scientist as toolsmith--Studies in interactive computer graphics. In Information Processing 77, Proceedings of IFIP Congress

77, 625-634.

Brooks, Jr., Frederick P. (1994).
The Computer Scientist as Toolsmith II. Keynote/Newell Award address at SIGGRAPH '94. Proc. of SIGGRAPH '94, 28, 4: 281-287.

Buchenau, Marion & Suri, Jane F. (2000).
Experience prototyping. In Proceedings of the 3rd Conference on Designing interactive Systems. DIS '00. ACM, New York, NY, 424-433.

Bush, Vannevar (1945).
As We May Think. In The Atlantic Monthly, July 1945. Available online from, e.g., http://www.theatlantic.com/doc/194507/bush.

Buxton, Bill (2007).
Sketching User Experiences: Getting the Design Right and the Right Design. Morgan Kaufmann.

Cook, Damon J. & Bailey, Brian P. (2005).
Designers' use of paper and the implications for informal tools. In ACM International Conference Proceeding Series, vol. 122. Computer-Human Interaction Special Interest Group (CHISIG) of Australia, 1-10.

Cooper, Alan (1999).
The Inmates are Running the Asylum. SAMS.

Csikszentmihalyi, Mihaly (1996).
Creativity: flow and the psychology of discovery and invention. New York: Harper Perennial.

Davis, Richard C., Colwell, Brien & Landay, James A. (2008).
K-Sketch: A "Kinetic" Sketch Pad for Novice Animators. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '08. ACM, New York, NY, 413-422. http://www.k-sketch.org

Dow, Steven, Saponas, T. Scott, Li, Yang, & Landay, James A. (2006).
External representations in ubiquitous computing design and the implications for design tools. In Proceedings of the 6th Conference on

Designing interactive Systems. DIS '06. ACM, New York, NY, 241-250.

Greenberg, Saul. (1993).
The Computer User as Toolsmith: the Use, Reuse, and Organization of Computer-Based Tools. Cambridge University Press.

Hartmann, Björn, Klemmer, Scott R., Bernstein, Micahel, Abdulla, Leith, Burr, Brendan, Robinson-Mosher, Avi & Gee, Jennifer (2006).
Reflective physical prototyping through integrated design, test, and analysis. In Proceedings of ACM Symposium on User interface Software and Technology. UIST '06. ACM, New York, NY, 299-308. http://hci.stanford.edu/dtools

Hartmann, Björn, Abdulla, Leith, Mittal, Manas & Klemmer, Scott R (2007).
Authoring sensor-based interactions by demonstration with direct manipulation and pattern recognition. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '07. ACM, New York, NY, 145-154. http://hci.stanford.edu/exemplar

Hartmann, Björn, Doorley, Scott & Klemmer, Scott (2008).
Hacking, Mashing, Gluing: Understanding Opportunistic Design. IEEE Pervasive Computing July-September 2008.

Hemmert, Fabian, Joost, Gesche, Knörig, André & Wettach, Reto (2008).
Dynamic knobs: shape change as a means of interaction on a mobile phone. In CHI '08 Extended Abstracts on Human Factors in Computing Systems. CHI '08. ACM, New York, NY, 2309-2314.

Hippel, Eric von (2005).
Democratizing Innovation. The MIT Press. Freely available online at http://web.mit.edu/evhippel/www/democ1.htm.

Kaptelinin, Victor & Nardi, Bonnie A. (2006).
Acting with Technology: Activity Theory and Interaction Design. MIT Press.

Klemmer, Scott R., Newman, Mark W., Farrell, Ryan, Bilezikjian, Mark, & Landay, James A. (2001).

The designers' outpost: a tangible interface for collaborative web site. In Proceedings of the 14th Annual ACM Symposium on User interface Software and Technology. UIST '01. ACM, New York, NY, 1-10. http://dub.washington.edu:2007/projects/outpost/

Knörig, André (2006).
Free the body and the mind will follow: An investigation into the role of the human body in creativity, and its application to HCI. Diploma Thesis, University of Applied Sciences Wedel, Germany. Available at http://andreknoerig.de/projects/free-the-body

Li, Yang & Landay, James A. (2005).
Informal Prototyping of Continuous Graphical Interactions by Demonstration. In Proceedings of the ACM Symposium on User Interface Software and Technology. UIST 2005. ACM, New York, NY, 221-230.

Li, Yang & Landay, James A. (2008).
Activity-based prototyping of ubicomp applications for long-lived, everyday human activities. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems. CHI '08. ACM, New York, NY, 1303-1312. http://activitystudio.sourceforge.net

Lin, James, Newman, Mark J., Hong, Jason I. & Landay, James A. (2000).
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. ACM, New York, NY, 510-517. http://dub.washington.edu:2007/denim

Löwgren, Jonas & Stolterman, Erik (2004).
Thoughtful interaction design: A design perspective on information technology. The MIT Press.

Löwgren, Jonas (2008).
Interaction Design. Retrieved 9 July 2008 from Interaction-Design.org: http://www.interaction-design.org/encyclopedia/interaction_design.html

Mellis, David, Banzi, Massimo, Cuartielles, David & Igoe, Tom (2007).
Arduino: An Open Electronic Prototyping Platform. Alt CHI 2007.
http://www.arduino.cc

Merleau-Ponty, Maurice (1945). *The Phenomenology of Perception*. English translation 1962. London, Routledge.

Moggridge, Bill (2006).
Designing Interactions. The MIT Press.

Myers, Brad A., Hudson, Scott E. & Pausch, Randy (2000).
Past, Present and Future of User Interface Software Tools. ACM Transactions on Computer Human Interaction. March, 2000. 7(1), 3-28.

Pugh, Stuart (1990).
Total Design: Integrated Methods for Successful Product Engineering. Addison-Wesley Publishing.

Reas, Casey & Fry, Ben (2007).
Processing: A Programming Handbook for Visual Designers and Artists. MIT Press. http://www.processing.org

Resnick, Mitch, Myers, Brad, Nakakoji, Kumiyo, Shneiderman, Ben, Pausch, Randy, Selker, Ted & Eisenberg, Mike (2005).
Design Principles for Tools to Support Creative Thinking. In NSF Workshop Report on Creativity Support Tools. Published online at http://www.cs.umd.edu/hcil/CST.

Schneider, Beat (2005).
Design - eine Einführung: Entwurf im sozialen, kulturellen und wirtschaftlichen Kontext. Birkhäuser Verlag für Architektur.

Schön, Donald (1983).
The reflective practitioner: how professionals think in action. New York: Basic Books.

Stolterman, Erik, McAtee, Jamie, Royer, David & Thandapani, Selvan

(2008)
Designerly Tools. Design Research Society Conference, Sheffield, 2008.

Svanæs, Dag (1999).
Understanding Interactivity: Steps to a Phenomenology of Human-Computer Interaction. PhD Dissertation. Dept. of Computer and Information Science, Norwegian University of Science and Technology. Trondheim, Norway.

Terry, Michael, Mynatt, Elizabeth D., Nakakoji, Kumiyo & Yamamoto, Yasuhiro (2004).
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. ACM, New York, NY, 711-718.

Winograd, Terry, Bennett, John, Young, Laura De & Hartfield, Brad (eds.) (1996). *Bringing Design to Software*. Addison-Wesley Publishing.

Without Authors:

Eleven lessons: Managing design in eleven global companies (2007). Design Council, UK. Retrieved from http://www.designcouncil.org.uk/Documents/About%20design/Eleven%20Lessons/PDF%20Eleven%20Lessons_complete_study.pdf

Eleven lessons: Managing design in eleven global companies: Desk research report (2007). Design Council, UK. Retrieved from http://www.designcouncil.org.uk/Documents/About%20design/Eleven%20Lessons/Desk%20Research%20Report.pdf

tool. (2008). In Encyclopædia Britannica. Retrieved July 07, 2008, from Encyclopædia Britannica Online: http://www.britannica.com/EBchecked/topic/599411/tool

LIST OF FIGURES