Jürgen Spangl

The Catalyst Kit

Encouraging collaboration and design thinking in interactive systems development

Doctoral thesis submitted in September 2008 at

Vienna University of Technology
Design and Assessment of Technology Institute
Vienna, Austria

Supervisor: Ass. Prof. Dipl.-Ing. Dr. techn. Peter Purgathofer


design
process
engineers
team
user
interaction
product
development
software
project
designers
experience
different
also
personas
within
scenarios
research
agile
specifications
designer
many
people
prototypes
way
projects
understanding
figure
boundary
one
can
time
even
good
knowledge
engineering
problem
client
used
information
internal
concepts
methods
use
users
technical
environments
role
tools
two
environment
following
stories
models
feedback
best
processes
approach
need
system
chapter
teams
describe
input
flows
practice
work
features
objects
plan
catalyst
often
still
involved
better
certain
everyone
without
just
sketches
ref
artifacts
point
needed
possible
actual
beginning
might
using
context
build
needs
make
described
usually
customer
get
selection
provide
approaches
important
interface
kit
view
usability
part
phase
fuse
concept
detailed
object
whole
method
much
problems
business
early
already
detail
three
based
tool
first
working
see
new
ideas
collaboration
able
members
perspective
driven
guide
setting
prototype
sketching
furthermore
therefore
sketch
model
cards
case
disciplines
issues
together
style
every

Previous page:
word cloud generated from a statistical analysis of the text of this dissertation.
http://wordle.net/

Abstract

A holistic view – including the product as whole and above all the human perspective within the social and environmental context – on software development is still not standard. Designers and software engineers are both experts in their respective specific field, yet only few methods and tools exist to build a common understanding of the overall product to support a smooth collaboration between them – from the first concepts to the actual product.

My thesis builds upon discussions on the theory and practice of software development in the perspective of design theory and upon the research on boundary objects – objects which are shared by both disciplines to empower collaboration. The experience acquired from many industry projects and case studies from the literature provide the basis for researching the collaboration between designers and engineers on software development projects from three different viewpoints: process, people, and artifacts.

Based on those findings I propose a toolkit – the Catalyst Kit – to encourage a holistic view on software development. The catalyst kit fosters design thinking by focusing on the collaboration within the team and by providing methods and tools to improve the transition from concepts to the final product. By treating software development as a dialog between disciplines we can understand each perspective separately from different points of view. Communicating the rationale of the methods, instruments and tools of each discipline supports better understanding and better team work, and leads to better products in the end.

Kurzfassung

Eine ganzheitliche Betrachtung – die das ganze Produkt, und vor allem auch die menschliche Perspektive innerhalb eines sozialen und umfeldbedingten Kontexts umfasst – ist nach wie vor nicht die Regel in der Softwareentwicklung. Designer und Softwareentwickler sind beide Spezialisten in ihren jeweiligen Fachbereichen, es existieren jedoch nur wenige Methoden und Werkzeuge, die ein gemeinsames Verständnis des gesamten Produkts fördern und eine reibungslose Zusammenarbeit unterstützen – beginnend mit der Konzeption bis hin zum fertigen Produkt.

Meine Dissertation stützt sich auf den Diskurs über die Theorie und Praxis der Softwareentwicklung unter Bezugnahme auf aktuelle Designtheorien und auf die Forschung über boundary objects – Objekte, die von beiden Disziplinen verwendet werden und dadurch die Zusammenarbeit stärken. Die Erfahrung aus vielen Kundenprojekten und Fallstudien aus der Literatur bilden die Grundlage für die Erforschung der Zusammenarbeit von Designern und Ingenieuren in Softwareentwicklungsprojekten aus drei Perspektiven: jener des Prozesses, jener der Menschen und jener der Artefakte.

Basierend auf diesen Erkenntnissen schlage ich den Catalyst Kit vor – ein Toolkit, das eine ganzheitliche Sicht auf Softwareentwicklung unterstützt. Der Catalyst Kit fördert design thinking durch eine Fokussierung auf die Zusammenarbeit des Teams und durch die Bereitstellung von Methoden und Werkzeugen zur Verbesserung des Übergangs von Konzepten bis hin zum Endprodukt. Wenn man Softwareentwicklung als Dialog der beteiligten Disziplinen sieht, kann man die Perspektive jedes einzelnen von den Standpunkten der anderen aus verstehen. Das Erklären und Begründen der verwendeten Methoden, Instrumente und Werkzeuge der jeweiligen Disziplin verbessert das Verständnis und fördert die Teamarbeit – und führt so zu besseren Produkten.

Acknowledgments

Karin, without her I would not have been able to finish my thesis. Her belief in my ability to get started at the very right moment gave me the strength to actually get started and finish it. Thank you Karin for always being there when I needed you.

Peter Purgathofer, my thesis advisor, thank you for many interesting discussions on design theory and practice and for helping me to find my bearings again when drifting too far away from the actual subject.

Special thanks to Tom Haberfellner, who not only explored with me the ups and downs of entrepreneurship, but was and is – with his inspiring ideas, approaches and concepts and his ability to give and take critique – an excellent sparring partner in many projects and for this thesis.

The team of GP designpartners, thanks for being a great team in numerous projects and creating such a wonderful design environment.

Thanks to the members of dshg – our study group – namely Karin Kappel, Jörg Summer and Martin Tomitsch. Getting your input and feedback and discussing our subjects and different kinds of issues in writing our theses was always a pleasure.

Last but not least, thanks to all proofreaders for assisting me in editing the final version, and to all my friends and fellows for their input and feedback.

Author's note

Throughout the thesis I use *we* when talking about the experience I shared with the interaction design teams on the many different projects. I use *I* for referring to personal experiences, e.g. design rationale, and for my own thoughts and conclusions.

Contents (short)

Contents

Part 1

Introduction to the research area

The objective of this research is to encourage collaboration and design thinking in interactive systems development. The aim of Part 1 is to introduce the research problem, its objectives and how the research has been approached.

Software design as understood today is usually only concerned with the technical architecture of the software (the internal design of software). Many have asked for a more holistic view on software design, to include the whole product and especially the human perspective within a social and environmental context (Brooks 1987, Buxton 2007, Floyd 1992a, Floyd 1992b, Winograd et al. 1996). Some are going even as far as understanding software development as a design discipline of its own (Purgathofer 2006). But following a holistic design thinking approach is rather difficult in practice. So far no adequate methods or frameworks exist.

My experience on many industry projects over the last ten years is such that the job of an interaction designer is twofold: in our first role we act as the *attorney of the users* (Doctorow 2008, p 245) and come up with adequate concepts, but our second – and at least equally important – role is to foster communication between the users and our clients on the one hand and between the different departments within our clients (mainly marketing or product development and software engineering) on the other hand to get actual user-centered concepts built. Much literature is published on methods, tools and approaches for interaction design and user-centered design (e.g. Cooper & Reimann 2003, Cooper 2004, Preece et al. 2002, Saffer 2006, Shneiderman & Plaisant 2004) but little on how to integrate this design thinking approach into software development

(Ashley & Desmond 2005, Blevis et al. 2006, Folmer et al. 2006). Even if Bill Buxton's (2005) postulation for a *Chief Design Officer* would come true, it is certain that it would be a lot easier to develop good user-centered software, but still the team members would need tools and methods to implement such a demand.

The curricula in computer science are still driven by an engineering approach rather than by a design approach, such as is the discipline of architecture (Purgathofer 2003, Purgathofer 2006). Therefore even if this would change within the next few years, many years would need to come to change this in the industry. What does happen is that the industry is more and more focusing on the users' needs, or to say it differently, is focusing on the *user experience* of the product and the corresponding services. Sadly currently this is only the case with job titles or within the public relation of the companies (Arnowitz & Dykstra-Erickson 2005a, Berkun 2005, p 48). Some interaction designers and usability engineers at our local face-to-face *IxDA*-meeting (Spangl & Haberfellner 2004) complain that their job mostly consists of writing user documentation or of something else with little impact on the projects. This reality is also supported by hard facts, such as the designer-to-developer ratio. Within Microsoft there are 42 developers per a single designer, and this is already a good ratio, with the usual ratio being 100 developers to 1 designer (Vanka 2007). I do not argue that it should be 1:1 or even 1:10, but such a low ratio does transport the current state of user-centered design and design thinking in general within software development.

I believe a change can only happen if supported by the whole team. In my thesis I therefore propose a toolkit – the *Catalyst Kit* – to (1) foster design thinking by focusing on the collaboration within the team and (2) provide methods for improving the transition from concepts to the final product. For this to happen the toolkit itself cannot be rigid. It has to give room for tinkering and shaping the toolkit and the single entities of the toolkit – the catalysts – to the needs of the actual project and environment. Furthermore I will provide best practices of our experience to show which catalysts work best within plan-driven and agile environments.

User experience (UX)
[is] concerned with improving the design of anything people experience: a web site, a toy, or a museum. UX is inherently interdisciplinary, synthesizing methods, techniques, and wisdom from many fields, ranging from brand design to ethnography to library science to architecture and more.
— Quesenbery et al. 2005

IxDA
The Interaction Design Association (IxDA) is a member-supported organization committed to serving the needs of the international interaction design community.
http://www.ixda.org

1 Scope of this thesis

1.1 Motivation

The scope of this thesis is driven by two statements. The first one is by Cooper:

> *[…] good, even great, design is meaningless unless it gets built.*
> — Cooper 2004, p 226

The second one is by Buxton on software development:

> *Neither design nor engineering is sufficient for the task at hand, and both are essential. What is required is a new relationship involving an adaptation of both skill sets that reflects the demands of the new design challenges.* — Buxton 2007, p 15

Designers and software engineers are specialists in their respective area of expertise, but they usually have very little training in actually working together. Only few methods and tools exist to build a common understanding of the overall product to support a smooth collaboration starting from concepts to the actual product. The core of this dissertation is to research and explore both methods and tools which encourage the collaboration among designers and engineers.

1.2 Boundaries

Calling for a more holistic view on software production should include all perspectives involved: the user, business, and engineering perspective. I try to take different viewpoints in researching this subject, nevertheless I have to admit that I cannot deny a certain bias towards a design perspective. It would be interesting to see catalysts cards developed by business people and engineers. But with the open design of the catalyst kit, I hope this will happen in the course of using it.

The thesis will focus mainly on the collaboration between designers and engineers, and only occasionally touch the business side. This is because of three reasons: (1) I believe that the collaboration between designers and engineers is especially crucial to build user-centered products. (2) Including also the business side would have opened a whole different subject. While I am very interested in this area, one has only a certain amount of time to dedicate to research. This gives room for future research. (3) My background in computer science (engineering) and my passion and profession in design.

1.3 Contribution

The objective of this thesis is to contribute to the research of software engineering and design following Erik Stolterman's understanding of design research:

> *Based on a comparison between the notion of complexity in science and in design, it is argued that science is not the best place to look for approaches and methods on how to approach design complexity. Instead, the case is made that any attempt by interaction design research to produce outcomes aimed at supporting design practice must be grounded in a fundamental understanding of the nature of design practice.*
> — Stolterman 2008

I accomplish this by:

- An overview on how designers and engineers collaborate on software development projects from different points of view, supported by my experience of design practice in industry projects – and case studies from the literature.
- The catalyst kit – a toolkit to enhance the collaboration between different communities of practice, namely designers, engineers and business people in projects. By treating software development as a dialog between disciplines we can understand each perspective separately from different points of view. Communicating the rationale of the methods, instruments, tools of each discipline supports better understanding and better team work, leading to *better products* in the end.

Best product
To our dismay, the »best« product isn't the one with the best interface or even the one that best serves the users' purpose. »Best« can be first to market, coolest, prettiest, fixes the worst previous problems, ships with the least effort, etc. – defining »best« is infinitely variable.
— Arnowitz & Dykstra-Erickson 2005b

Despite the truth of this quote, I still believe that ultimately the best product is the one which provides the best user experience. Therefore when I use the term *best* or *successful product* in this thesis it is in regard to user experience.

2 Outline of the thesis

The dissertation is structured into five parts:

- **This part** covers the motivation, scope and the research approach of my thesis.
- In **part 2** I will set the stage for this research by providing definitions for design and software engineering and by elaborating on the two major influences of this thesis: Peter Purgathofer's habilitation treatise *designlehren: zur gestaltung interaktiver systeme* (Purgathofer 2003) – which discusses the theory and practice of software development in the perspective of design theory – and boundary objects (objects which are shared by both disciplines to

empower collaboration). Furthermore, I give an introduction to the projects to which I refer throughout this dissertation.

- **Part 3** forms the main part, in which I will investigate the collaboration between designers and engineers on software development projects from three different points of view: process, people, and artifacts.
- Finally, **part 4** introduces the catalyst kit – a toolkit built on the findings of part 3 – as a way to strengthen the collaboration in software development projects.
- The conclusions of this research and an outlook on possible future research in this field are discussed in **part 5**.

3 Research approach

This research belongs to the class of applied research in the field of interaction design research aimed at supporting interaction design and software engineering practice.

3.1 Theoretical goals

Methods exist to describe the software life cycle, be it in plan-driven or in agile environments. For plan-driven environments many have written in great detail on the different phases of the process, especially on the software engineering part. Also the agile methods focus mainly on the engineering or implementation part. Even with design only getting recently more attention within the Human Computer Interaction (HCI) discipline, in the last years many papers and books have been published covering *User-centered Design* (UCD) methods and processes for software development. However, methods and processes for a holistic approach for software development are sparse. As such the theoretical goals of this thesis are to further develop existing methods and theories.

User-centered Design (UCD)
The philosophy of user-centered design sets the focus on the users' needs, wants and limitations, or simply put: *users know best* (Saffer 2006, p 31). Ideally the user is directly involved in every stage of the design process.

3.2 Practical goals

The practical goals of this thesis are to transform the experiences from industry projects combined with existing theories into a toolkit, which supports teams by getting a better understanding of the different disciplines and their approaches involved in a software project.

3.3 Research methods

The aim of this research is the practical applicability of its results. Consistency between theoretical models and observations can be reached by a dual scientific approach. Jörgensen (1992 in Karlsson 2002) describes a model, which combines both the practical, problem based and the theory based research (figure 1). My experience throughout several industry projects forms the problem based research of this thesis.

The process of applied research is described by Jörgensen (1992) as an ongoing iterative process of analysis and synthesis. The findings of the observation and analysis of existing approaches can be directly applied in practice. The theoretical research covers the development of new methods by the scientific community. The newly produced results can also directly be applied to practical applications. Within applied research analysis and synthesis are constantly influencing each other.

3.4 Research activities

The research for this thesis is conducted through different kinds of activities: literature studies, reflections on ten years of experience in the industry, and experiences from teaching interaction design at two universities.

Literature

The literature studies cover design theory, design processes and methods, user-centered design, software engineering, software design, systems design, software life cycle processes and project management.

Experience of ten years working in the industry

The most important contribution for this thesis comes from my experience of working as an interaction designer and project manager over the last ten years. My interest in design and a holistic view on software development was established and deepened at the uid-lab at the Institute of Design and Assessment of Technology led by Peter Purgathofer. At Razorfish (one of the then leading multimedia companies with offices around the globe) I gathered the experience of working within larger teams and with large corporate clients. In my current position as a managing partner and interaction designer at GP designpartners (a product and interaction design agency based in Vienna) my role in projects is mostly that of a catalyst between design and engineering. Furthermore at GP I gathered valuable insights into the processes and methods of product design, which also inspired some of the developed methods.

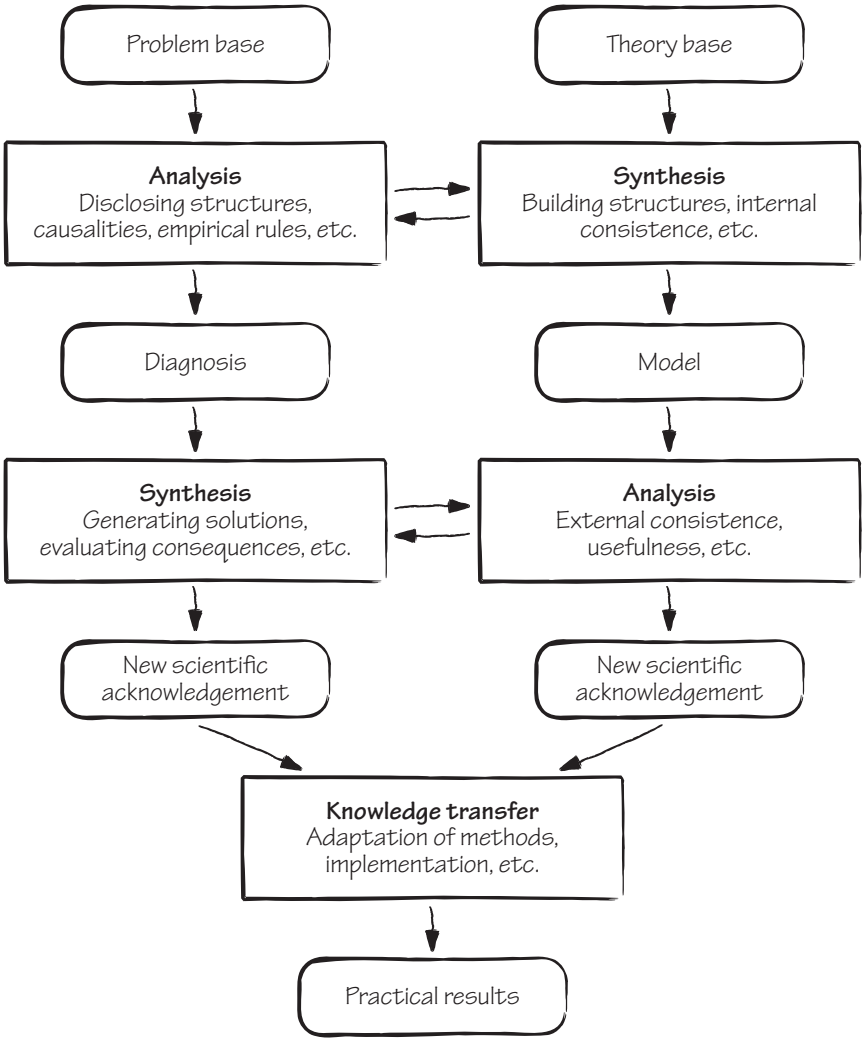


Figure 1: A method for applied research with a focus on the interaction between theory and practice (Jörgensen 1992 in Karlsson 2002).

Teaching

Methods and approaches described in the thesis have been introduced and discussed over the last three years in theoretical and practical lectures. At the University of Applied Sciences Vienna in lectures on Software-Systems and Human Interfaces, Human Computer Interaction, and Application Development. At the University of Applied Sciences Burgenland in lectures on User Interface Design and Usability.

Part 2

Setting the stage

Part 2 provides the theoretical basis with definitions for design and software engineering, a discussion on the theory and practice of software development in the perspective of design theory, and an overview of the research on boundary objects.

The traditional software development process follows a standard engineering approach, in which the design of the software is usually separated from the implementation. Löwgren (1995) calls those two parts *internal design* and *external design*. Internal design is the part which is usually framed as software design in the literature, laying out a plan for implementing the software from a given set of requirements. External design on the other hand is the design of the specific part of the software which interacts with the user. Bringing a discussion of Simon (1996) on the design in the context of artificial fields into software design, Tayler and van der Hoek also separate it at the same line as Löwgren and call it according to Simon (1996) the *outer environments* and *inner environments* (Taylor & van der Hoek 2007).

Most of the current research in software development focuses either exclusively on external design (e.g. Cooper 2004, Cooper & Reimann 2003, Preece et al. 2002, Saffer 2006, Shneiderman & Plaisant 2004) or exclusively on internal design (e.g. Sommerville 2006). Only recently a few workshops took place on the subject of bridging the gap between human computer interaction (external design) and software engineering (internal design) (Harning & Vanderdonckt 2003, Harning et al. 2005, John et al. 2004, Kazman et al. 2003, Kazman et al. 2004). Apart from position papers on the status quo and the differences of human computer interaction and software engineering (e.g. Belenguer et al. 2003, Law 2003, Müller-Prove 2003, Nunes 2003) various contributions from different viewpoints

with the aim of closing the gap were discussed. Many proposed process models to better integrate human computer interaction into existing software engineering processes (e.g. Constantine et al. 2003, Jespersen & Linvald 2003, Jones et al. 2004), especially for the unified process (e.g. Palanque & Bastide 2003, Silva & Paton 2003, Sousa & Furtado 2003a, Sousa & Furtado 2003b, Sousa & Furtado 2004). Bringing in usability earlier in the process – already while developing the software architecture, and not only at the end for the purpose of quality assurance – was another concern (e.g. Folmer et al. 2003, Folmer & Bosch 2004, Ionita et al. 2003, Juzgado et al. 2003). Others searched for similarities in both disciplines and adapted those methods accordingly to close the gap (e.g. Ferré 2003, Kerkow et al. 2005, Paech & Kohler 2003), e.g. by light weight evaluations (Gellner & Forbrig 2003) or pattern based usability inspection methods (Schmettow 2005) for software engineers. Teaching practices and methods from human computer interaction in software engineering curricula and vice versa was seen by many participants as a step in the right direction (e.g. Clemmensen & Nørbjerg 2003, Ludi 2003, Milewski 2003, Seffah 2004, Willshire 2003) to bring both disciplines together.

A workshop at the CHI 2004, the premier international conference for human computer interaction, in Vienna on *Identifying gaps between HCI, software engineering, and design, and boundary objects to bridge them* (John et al. 2004) brought my attention to boundary objects – objects which are shared by both disciplines to empower collaboration. As boundary objects are the foundation of my research they are covered in more detail in chapter 6 (p 19).

Interestingly none of the workshop contributions discussed the subject from a design theory or a designerly point of view. A reason for this might be that design is only recently considered in human computer interaction: *Slowly, and not without reluctance, CHI [Computer Human Interaction] has loosened its ties to psychological theory and engineering, and schooled itself in design* (Grudin 2006). Stolterman (2008) argues that *one reason why HCI research (aimed at supporting design practice) has not (always) been successful is that it has not been grounded in and guided by a sufficient understanding and acceptance of the nature of design practice*. Hence for design practice it is beneficial to research the gap of design and software engineering.

Purgathofer's habilitation treatise *designlehren: zur gestaltung interaktiver systeme* (Purgathofer 2003) discusses the theory and practice of software development in the perspective of design theory and is supported by experiences in design practice. His work, which is based on Gedenryd's (1998) and Lawson's (1997) publications, influenced my approach to software development and interaction design and forms the basis for this thesis.

4 Definitions

Software engineering and design have in common that both lack a generally accepted definition. Following are outlines of how software engineering and design are understood rather then defining the terms.

4.1 Software engineering

Software engineering is a rather young discipline. The term was coined in 1968 by Friedrich Bauer at the NATO conference (Bauer 1993) to address the perceived crisis in software development at this time. Until today there is no generally accepted definition for software engineering. Fortner (2007, pp 12–15) gives a good historical overview about the common definitions for software engineering in his masters thesis. I will use Boehm's definition for software engineering based on the general definition of engineering by *Merriam-Webster's Online Dictionary* (2008):

> *The application of science and mathematics by which the properties of software are made useful to people.*
> — Boehm 2006, p 12

He points out that with the phrase *useful to people* other sciences – such as behavioral sciences, management sciences, and economics, as well as computer science – are included within software engineering.

As much as I like this definition, currently it can only be seen as an idealistic definition. By looking at the history of software engineering the part *made useful to people* is only fulfilled rudimentary at best.

4.2 Design

Design, on the other hand, is very old, as old as products. But it took up to the sixties of the last century until the scientific research started with the first conference on the methodology of design hosted by the British Design Research Society and the American Design Methods Group. Purgathofer (2003, pp ix-xi) provides an assorted list of different definitions, and singles out one definition which describes best the never ending search for a definition:

> *The answer [to the question of what design is] is probably that we shall never really find a single satisfactory definition but that the searching is probably much more important than the finding.* — Lawson 1997 (1st Ed 1980), p 31

I would like to give another, more definitive, description of design by Charles Eames in a conversation with Madame Amic, which for Moggridge is peculiarly relevant to interaction design:

> *Q [Madame Amic]: What is your definition of »Design?«*
> *A [Charles Eames]: A plan for arranging elements in such a way as to best accomplish a particular purpose.*
>
> *Q: Is design an expression of art (an art form)?*
> *A: The design is an expression of the purpose. It may (if it is good enough) later be judged as art.*
>
> *Q: Is design a craft for industrial purposes?*
> *A: No – but design may be a solution to some industrial problems.*
>
> *Q: What are the boundaries of design?*
> *A: What are the boundaries of problems?*
>
> *Q: Does the creation of design admit constraint?*
> *A: Design depends largely on constraints.*
>
> *Q: What constraints?*
> *A: The sum of all constraints. Here is one of the few effective keys to the design problem – the ability of the designer to recognize as many of the constraints as possible – his willingness and enthusiasm for working within these constraints – the constraints of price, of size, of strength, balance, of surface, of time etc.; each problem has its own peculiar list.*
>
> *Q: Does design obey laws?*
> *A: Aren't constraints enough?*
> — Moggridge 2006, pp 648–9

Using such a design approach is often described as *design thinking* or a *designerly way of thinking* (Stolterman 2008).

5 Three generations of design models

Illustrating the current approach of software engineering and its critique can be accomplished nicely with the aid of Purgathofer's description of three generations of design models (Purgathofer 2003, pp 22–54). Design models describe a certain understanding or philosophy of design, explain

inherent methods and processes and state characteristics of circumstances.

5.1 Generation one: design as problem solving

The first generation is based on *problem solving*. Design is the rational process for finding the right – or the optimal solution – for a given problem. This is usually done in a chain of transformations from an abstract form (requirements) into a specification of what should be done (Löwgren 1995). In this model it is considered that the actual design is mental work. This structured approach tries to make the design independent from any performing person. Terms used for the methods of the first generation are *process models* (Purgathofer 2003, p 26), *engineering design* (Löwgren 1995), or *design as planning* (Gedenryd 1998, p 50 based on Jeffries et al. 1981).

The critical issue with this model is that there are hardly ever settings in which the problem is clearly stated and stable. As Parnas & Clements (1986) put it: *Determining the detailed requirements may well be the most difficult part of the software design process.* Ever changing requirements count among the biggest inconveniences on projects, but are a reality (Purgathofer 2006). Hence defining the problem itself becomes the most important part of this model.

5.2 Generation two: wicked problems

The second generation questions the assumptions of the first generations. It argues that the problem only emerges in conjunction with design. Furthermore, design problems are not as well defined as for example solving an equation or a puzzle. Rittel & Webber (1973) call them *wicked problems* and characterize them by ten attributes:

1. *There is no definitive formulation of a wicked problem.*
2. *Wicked problems have no stopping rule.*
3. *Solutions to wicked problems are not true-or-false but good-or-bad.*
4. *There is no immediate and no ultimate test of a solution to a wicked problem.*
5. *Every implemented solution to a wicked problem has consequences.*
6. *Wicked problems do not have a well-described set of potential solutions.*
7. *Every wicked problem is essentially unique.*

8. *Every wicked problem can be considered a symptom of another problem.*
9. *The causes of a wicked problem can be explained in numerous ways.*
10. *The planner (designer) has no right to be wrong.*

— Rittel & Webber 1973

According to McPhee (1996) problems become wicked when the human actions are the focus of the activity, and thereby are hard to conceive and solve.

> *By situating a design in the world of people, the design is affected by the people and the people are affected by the design. The same design issues may be viewed differently by each of the individuals involved. Each divergent perspective may influence the progress of the design in different and unpredictable ways.*
> — McPhee 1996, p 10

How can one tackle those problems, which cannot be stated in a stable way? Rittel & Webber (1973) suggest with a discussion between the problem and the solution. If a misunderstanding in the discussion arises, it is framed as a question. The question then turns the argument into an inquiry. Through this approach a dialog is started with the chance to discover possible solutions. Analysis and synthesis are hereby performed jointly within one activity, instead of planing first and then implementing. Analysis is possible through synthesis. By trying different things new questions and insights emerge. Dealing with the new findings acts as catalyst in the joint process of analysis and synthesis.

The discussion process is – in the tradition of problem solving – mental work. Generation two models are concerned with the characteristics of design circumstances and can therefore be called *structural models* rather than *process models* (Purgathofer 2003, p 24).

Purgathofer states that Rittel & Webber's model is still based on a problem existing a priori, but that in the design practice dealing with the problem itself (in an analytical way) is inferior to the synthesis – the design of the product. He argues that synthesis should be the heart of design models.

5.3 Generation three: doing for the sake of knowing

In generation three the discussion process changes from a mental task to an action based task. Design only happens by interacting with the context, with design tools and instruments. The problem itself fades even

more into the background than in generation two. Concepts such as *doing for the sake of knowing*, *problem setting*, *primary generator* or *inquiring materials* take on greater significance. Action and interaction with reality becomes more important.

Doing for the sake of knowing

Doing or the sake of knowing can be described best by the habit to set actions only to gain knowledge about a certain situation:

> *The rudimentary prototype of experimental doing for the sake of knowing is found in ordinary procedures. Then we are trying to make out the nature of a confused and unfamiliar object, we perform various acts with a view to establishing a new relationship to it, such as will bring to light qualities which will aid in understanding it. We turn it over, bring it into a better light, rattle and shake it, thump, push and press it, and so on. The object as it is experienced prior to the introduction of these changes baffles us; the intent of these acts is to make changes which will elicit some previously unperceived qualities, and by varying conditions of perception shake loose some property which as it stands blinds or misleads us.* — Dewey 1999, p 87 cited in Gedenryd 1998, p 122

Gedenryd distinguishes between two kinds of inquiring actions: exploration and experimentation. In the quote above Dewey talks about exploration, the act of discovering new facets of something in an interactive process. A vivid example is the computer game *Tetris*. The player rotates and moves the brick to explore where it might fit, instead of just looking at it and considering the best solution (Kirsh & Maglio 1992).

Experimentation, on the other hand, is more powerful than exploration. Experimentation has a certain goal in mind and tries to evaluate it. Instead of mentally simulating ideas with if-then-scenarios, they are tested with experiments in reality. For designers sketching is the primary method for experimentation. They sketch to test the consequences of their ideas and to generate new ones. For more details on sketching and the characteristics of a sketch see *Digression: Sketches* (p 90).

Problem setting

Donald Schön (1983) coined the term *design as problem setting* in his model of design. He describes a dialog between an architect, Quist, and his student, Petra, who runs into a deadlock searching for a solution for a design project. Quist reveals that the reason for her stalemate is not the actual design situation itself but has its origin in her own decisions.

Instead of offering a solution to the design problem, he suggests to *reframe* the problem into a different one. Schön calls this *problem setting*.

Here, problem setting is used as an instrument, as Gedenryd points out:

> *He [the architect] thereby treats the framing of the problem as an instrument to adapt to the purpose at hand, here, to yield a good design solution. In the instrumental view, also the problem itself has a purpose. The problem statement serves to specify the function of the design solution. Hence, when the problem is not regarded as something given, it is also no longer a hindrance making life difficult for the designer. Instead, it serves to spell out the purpose of the eventual design. And the activity of problem setting becomes an inquiry into this purpose, in order to understand what it is. Thus also the task of problem setting makes a contribution to the designer's understanding.*
> — Gedenryd 1998, p 83

Problem setting together with doing for the sake of knowing are major inseparable drivers for the design process. Purgathofer (2003) interprets the above dialog in this regard as follows: testing (doing for the sake of knowing) her concept (problem setting) led Petra into an insoluble problem – at least for her. Quist adapted her assumptions (problem setting) and showed Petra over the course of the dialog with sketches and explanation (doing for the sake of knowing) how she can avoid her problems.

Purgathofer points out another interesting characteristic of problem setting: the fact that Petra was not able to reframe the problem by herself indicates her strong – emotional and factual – involvement in developing the concept and hereby an impediment in reframing it. Apparently, it is impossible for Petra to recognize that her initial concept is the root cause for her problem.

Primary generators

In a research on the motivation and intention of architects in their design process Darke (1978) discovered that architects are inclined to come up with a basic concept or idea already early on. They do this to be able to tackle the complexity and uncertainty of the design problem. Darke (1978) calls those early ideas *primary generators*. She proposes a design model, which turns the analysis-synthesis-model upside down – the primary generator follows a phase in which the concept is further developed and then analytical researched – thus synthesis-analysis. Purgathofer (2003) points out that in practice this model is more valid than the model

of generation one *design as problem solving*, nevertheless it is only a part of the understanding of design models of generation three. But it shows that designers come up with decisions already before substantial informations are available about the design circumstances. Purgathofer states that the *primary generator* is the first and most influential *problem setting* and thus calls it *generator concept* (Purgathofer 2003, p 50). With the generator concept essential characteristics of the project are predefined early on.

An attribute of the primary generator is that it is often not challenged by the designer later on. This is the case in the problem of Petra stated above: she does not question her generator concept and thus is unable to recognize that it is the very reason for getting stuck in this project. Purgathofer points out a study of Rowe, which shows this pitfall:

> *Another aspect of design thinking evident in the foregoing design studies is the tenacity with which designers will cling to major design ideas and themes in the face of what, at times, might seem insurmountable odds. […] Even when severe problems are encountered, a considerable effort is made to make the initial idea work, rather than to stand back and adopt a fresh point of departure.* — Meyer 1989 cited in Purgathofer 2003, p 51

Lawson describes a reason for this phenomenon:

> *[…] early anchors can be reassuring and if the designer succeeds in overcoming such difficulties and the original ideas were good, we are quite likely to recognize this as an act of great creativity.* — Lawson 1980 cited in Purgathofer 2003, p 51

Inquiring materials

Based on doing for the sake of knowing Gedenryd describes a model of *interactive cognition*, which extends the usual view of thinking as *intramental cognition* (Gedenryd 1998). His theory describes thinking as an interaction between the individual and the world:

> *Interactive cognition is meant to indicate that mind, action and world mutually determine an individual's doings, in interaction. […] In interactive cognition, the cognizing individual on the one hand, and the world on the other, reciprocally influence each other. In other words, mind and world interactively determine cognitive performance.* — Gedenryd 1998, p 111

Figure 2 shows both models side by side.

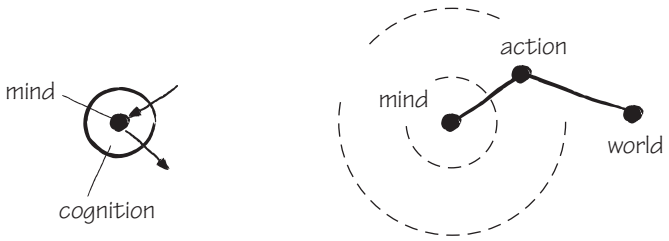


Figure 2: Simplified schemes of the traditional, sharply delimited view of cognition as an intramental process (on the left), and the wider and less distinctly circumscribed view of cognition adopted in the *interactive cognition* (on the right) (Gedenryd 1998, p 12).

Gedenryd (1998, p 149) defines inquiring materials as *working materials with a cognitive purpose*. Compared to doing for the sake of knowing – where the action itself has a cognitive purpose – here the produced material itself provides the cognitive purpose. E.g. this means for sketches, that the act of sketching has a cognitive purpose, as well as the sketch itself.

> *If a material is created in the design process, not as an end product, but rather to serve the inquiry that the design process is, then it has a* ***second, inquiring purpose****. […] this parallels the inquiring purpose of action. An »inquiring material« then, like and inquiring action, does not function as an* ***end*** *product of design, but as a* ***means for*** *the inquiry that design is.*
> — Gedenryd 1998, p 149

Typical artifacts of the design process, which act as inquiring materials, are sketches, prototypes, models, scenarios, etc. to name a few.

To conclude, software engineering is mostly driven by an understanding of design according to generation one. Most software engineering methodologies still build on the idea of having a given stable problem, which can be captured by requirements engineering, followed by finding the solution for it. Agile software engineering methods, such as extreme programming, acknowledge the unpredictability of software projects and propose a different approach. Although agile methods can be seen as being driven by a design understanding of generation three, open issues remain from a design point of view.

The chapter *Process* (p 32) covers those issues and discusses approaches on how to bring a more designerly way of thinking – design understanding in the sense of generation three – to currently used software engineering processes in more detail.

6 Boundary objects

Software projects are *communities of interests* with team members of at least two different *communities of practice* (users and engineers) (Marick 2003). Each team member has his specialized *knowledge of practice* and this makes collaboration across functional boundaries and the adaptation to the knowledge developed within another practice difficult (Carlile 2002, Gasson 2005). Carlile conducted ethnographic studies in product development environments (development of an on-board vapor recovery valve and the early design of a car) to understand the nature of knowledge and the resulting boundaries (Carlile 2002, Carlile 2004). The settings and collaboration within teams in product development is comparable to those found in software projects. Also product development usually follows a plan-driven process, the boundaries for collaboration and knowledge sharing apply accordingly to agile environments, which are used more and more in software development. The following situation of a design engineer and a manufacturing engineer shows the knowledge boundaries that are generated as each of them has to solve different problems within their practices:

Community of interests
are groups with different backgrounds (communities of practice) working together to frame and solve a (design) problem (Arias 2000, Fischer 2001).

Community of practice
The term *community of practice* was introduced by Lave and Wenger in 1991. A community of practice is a group of people with a common interest in a subject, who interact regularly and share knowledge and practices (Fischer 2001, Lave & Wenger 1991).

> *As the sole representative from manufacturing engineering at this early design phase, he could easily summarize his frustration to me: »they [the design engineers] don't realize that the OVRV [on-board vapor recovery valve], with its high part count and 3,000,000-a-year volume, is going to be a completely different beast to deal with.« Mick had already strongly expressed the necessity of going to subassemblies to make assembly and testing easier, but so far he hadn't gotten any significant design changes approved. At each successive design meeting, the only critical changes he noticed were ones that improved the valve's »functionality.«*
>
> *In this design review meeting, Mick's statements about the »awkwardness of the design« only seemed to be inflaming the tempers of the sales representative and design engineers on the team. For Vaughn, the head design engineer, the past challenges of packing all of the functional requirements of such a complex »3-in-1« valve into such a small space made the success of the »working prototype« and the current design a significant achievement. However, what increased the tension in the room for everyone was the announcement that the customer was holding firm on having test parts delivered in eight weeks. Mick kept up with his arguments about subassemblies, and he would later say to me, »what am I going to do, sit and wait for the*

> *production launch (of the OVRV) for all hell to break loose? I'll have more than egg on my face then.«* — Carlile 2002, p 443

This situation, with modifications on the discussed subject, could also be captured on a software project.

6.1 An integrative framework for managing knowledge across boundaries

Susan Star (1990) introduced the term *boundary objects* for communicating different kinds of knowledge across boundaries in different ways:

> *Boundary objects are objects that are both plastic enough to adapt to local needs and constraints of the several parties employing them, yet robust enough to maintain a common identity across sites. […] Like the blackboard, a boundary object »sits in the middle« of a group of actors with divergent viewpoints.* — Star 1990, p 46

A boundary object is an object shared by several different communities of practice, but can be used or viewed differently by each of them. They contain enough detail to be comprehensible by any group, without being required to fully understand the context of use of the other groups.

Based on Stars classification of boundary objects Carlile develops three approaches (*progressive complex boundaries*) to knowledge and how to move it across boundaries (Carlile 2002) and expands it by three *progressively complex processes* into a framework for managing knowledge across boundaries (Carlile 2004):

- **A syntactic or information-processing boundary – transferring knowledge:** In this approach the syntax has been specified and agreed to in advance across boundaries. A shared repository, taxonomy or lexicon exists, and assumes that people *understand meanings in the same way* (Gasson 2005). Hence information can be processed syntactically or transferred across boundaries.
- **A semantic or interpretive boundary – translating knowledge:** A common lexicon is always necessary, but it is not always sufficient. The problem gets more complex especially when novelty occurs. The semantic approach acknowledges that people interpret information in different ways, which makes collaboration across boundaries difficult. Communication is accomplished through translation and learning about differences and dependencies at the boundary

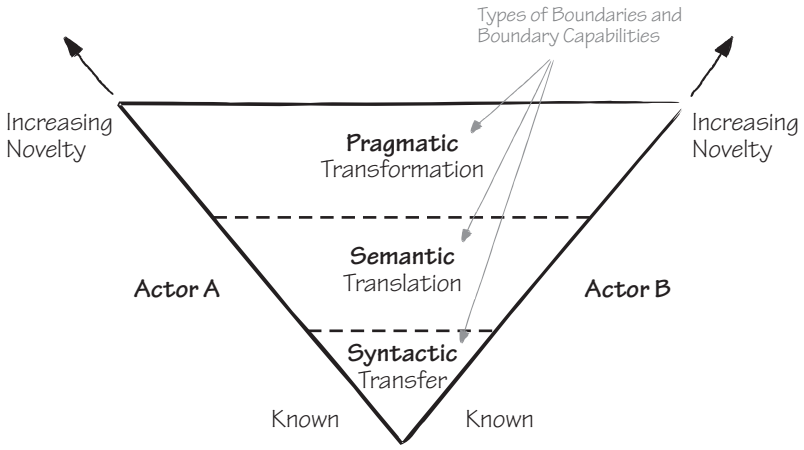


Figure 3: Integrated 3-T framework for managing knowledge across boundaries (Carlile 2004, p 558).

by use of processes and methods (e.g. standardized templates and forms, UML)

› **A pragmatic or political boundary – transforming knowledge:** The second approach assumes that all actors have the same interests, but this is not always the case. With conflicting interests knowledge within one community of practice leads to negative consequences in another one. Actors not only need to learn what is new, but sometimes also need to transform existing knowledge of e.g. the domain. Therefore the actors are not willing to make changes. Collaboration is accomplished by transforming both the domain-specific and the common knowledge by negotiating and learning about the consequences, differences and dependencies of the knowledge. Local domain specific knowledge is altered into a novel form of shared knowledge. Carlile (2002) combines for this boundary the third category (models and ideal-type objects, e.g. sketches, drawings, prototypes, mock ups) with the fourth category (terrain with coincident boundaries – maps, e.g. gantt charts, process maps, workflows) of boundary objects from Star (1990) into objects, models, and maps.

The three boundary types are shown in figure 3 in a hierarchical representation to show that the complexity increases with each level of boundary; and that each process or capacity of a boundary object requires the capacities of those below (Carlile 2004).

6.2 Characteristics of effective boundary objects

To illustrate the framework Carlile (2004) describes a case study of the development of a computational fluid dynamic (CFD) tool with a three-dimensional modeling technique and its combined use as a collaborative engineering tool instead of clay models in the early stages of product

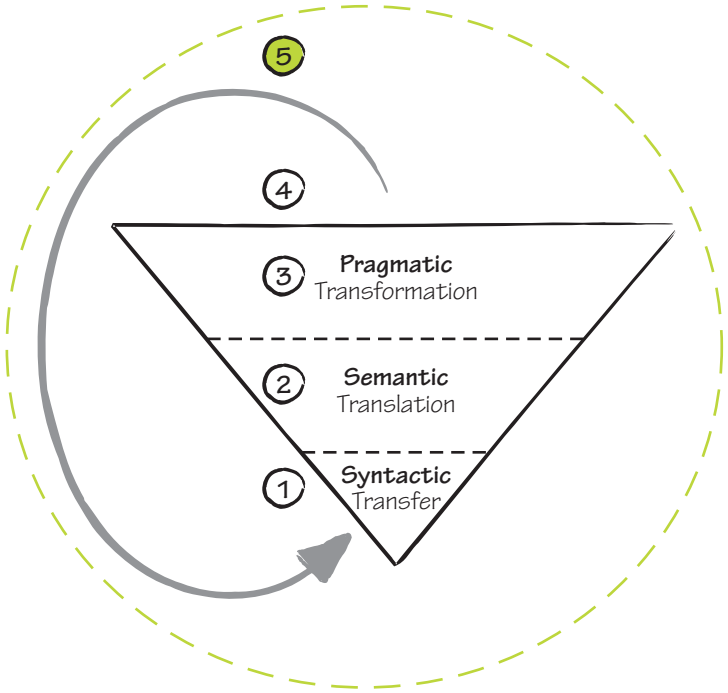
Firemen's tarp
A firemen's tarp was used in the early twentieth century to catch individuals jumping from burning buildings. [The lead designer] always stated that the effectiveness of the tarp came from three things. First, it had to be made of a strong material and fashioned in a way that allowed each fireman to easily hold and use it. Second, it needed to be held by several firemen pulling as hard as they could in different directions to safely break an individual's fall. And third, the firemen had to constantly look up at the individual and make adjustments to ensure the individual jumping landed safely in the middle of the tarp. — Carlile 2004, p 562

development within the auto industry. The tool was developed with input from all parties involved. The lead designer of the tool explained the purpose of the tool with the metaphor of a *firemen's tarp*: *the engineering tool would establish a shared way for each group to pull hard in their own direction and still make good trade-offs* (Carlile 2004, p 562). Developing the tool was not an easy task. Getting all teams involved and recognizing the benefits, even with giving up some of the control, was more challenging than technical issues.

The introduction of the tool was very effective in three out of four cases and led to a reduction of engineering time and costs. In the fourth case the costs were as high as previous redesign efforts. Based on Carlile (2002) and on the three cases Carlile (2004) identified four characteristics of effective boundary objects (figure 4):

1. Establishing a shared syntax or language for individuals to represent their knowledge. It has to present what is *at stake* for all parties. A shared syntax is an important feature for all three approaches of boundary objects.
2. Providing a concrete means for individuals to specify and learn about their differences and dependencies across a given boundary. With an effective method or process an individual can concretely specify what they know and what they worry about. This feature is important for both semantic and pragmatic boundary objects.
3. Facilitating a process in which individuals can jointly transform their knowledge. An actor must be able to transform the current approach into a cross-functional problem, otherwise his knowledge will have very little impact. Manipulation and altering of the content of a boundary object must be possible for individuals in order to apply their knowledge and transform the current knowledge at the boundary.
4. Supporting an iterative approach to allow actors to better identify which differences and dependencies have which consequence at the boundary.

So, what is then an effective boundary object in the context of software development? Personas (p 73), for example, are an effective boundary object, whereas the transcriptions of the user research are not, because the engineers are not able to set them into context. Screenshots alone would be an ineffective boundary object, because the behavior is not visible. But combined with flows (p 107) or explanations they become an effective boundary object.



5 Encourages a process of **discussing while building** the boundary object itself for establishing a buy-in of all members.

4 Supports an **iterative** approach where actors get better at developing an adequate common knowledge for sharing and assessing each other's knowledge.

3 A pragmatic capacity establishes common interests for making trade-offs and **transforming** domain-specific knowledge.

2 A semantic capacity develops common meanings for identifying novel differences and dependencies and **translating** domain-specific knowledge.

1 A syntactic capacity requires the development of a common lexicon for **transferring** domain-specific knowledge.

Figure 4: 3-T Framework with five characteristics: four characteristics of a pragmatic boundary capability by Carlile (2004, p 563), additionally I suggest a fifth characteristic: the process of discussing while building the boundary object itself.

6.3 Boundaries of boundary objects

In one of the four cases described by Carlile (2004) the tool was not successfully used. The reason for this was the carry-over of knowledge without questioning it. Boundary objects do not work equally well in all settings. Some work out in one project, but are not useful or become even show-stoppers in others. Boundary objects are no *magic bullets* (Carlile 2002, p 452). Problems and people change, thus it is hard to sustain the characteristics of boundary objects.

Not all knowledge can be presented at the same time. Carlile (2002) describes this with an analogy of a funnel: *Using a funnel analogy as a way of thinking about the product development process, by the time a downstream function (i.e., manufacturing engineer, production manager) begins to understand how upstream decisions make their objects and ends more difficult to problem solve, it is hard for a downstream function to make changes to that upstream knowledge.* This temporary dimension of dependency puts the upstream knowledge in a stronger political position relative to the downstream knowledge. Hence actors involved at a boundary do not occupy equal politically positions. This is a reason why the clay model was the dominant method used in the auto industry into the 1990s, *it represented what was at stake for the most upstream and powerful group, vehicle styling and their marketing champion* (Carlile 2004). On software projects it is the other way around. Engineers are positioned more downstream, but in most projects they have a more powerful position. Based on my experience the reason for this is that in software projects the engineers are positioned more upstream in a political sense: engineers

Magic bullet
also known as *silver bullet* is a term widely used for a effective solution to a difficult or formerly insoluble problem (*Merriam-Webster's Online Dictionary* (2008)). In software engineering the term gained fame by the paper *No Silver Bullet - Essence and Accidents of Software Engineering* by Brooks (1987).

as project managers, engineering driven processes, weak acceptance of design in software projects. So even if design, as a process phase, comes prior to engineering it is weaker positioned because the overall view of a software project is engineering driven, and not seen in a holistic way, with both seen as being equal.

6.4 Boundary objects in the context of this thesis

Boundary objects are an inspiration for the proposed catalysts (p 125). Many of the catalysts help to bridge semantic and pragmatic boundaries; though it is not a requirement for the catalysts to fulfill the definition of a boundary object.

My interpretation of the failed usage of boundary objects in one of the four projects is that in this case the boundary object was accepted as given, and the process of establishing it was missing. The establishment of negotiating the boundary object is important for its success. I suggest a *fifth characteristic* for effective boundary objects: *the process of discussing while building the boundary object itself* (figure 4). My proposed catalyst set (p 121) supports and encourages team members to discuss the current project situation and to manifest given and to discover new boundary objects.

7 Projects

In addition to case studies from the literature I will refer to the experience on projects I was involved in. This chapter gives a short introduction and summary to those projects referred throughout the thesis. Relevant aspects of the projects are described in detail within the appropriate topic. The projects span a broad spectrum of my experience within the last ten years in industry projects at the Vienna University of Technology, Razorfish and GP designpartners. Out of many projects I picked four to present a comprehensive view on different kinds of projects: small and large project teams; plan-driven and agile processes; a website, a software-application, a web-application and a physical product. All of the presented projects followed a user-centered design approach.

The client names and details of some of the projects have been modified to protect confidentiality.

7.1 fuse

In 2004 a med-tech company and global leader in prosthetics asked GP designpartners to develop both the interaction design and a user interface style guide to combine their range of seven independent ap-

Figure 5: Four screenshots of fuse (from top left to bottom right): job specifications for entering detailed measurements; setting reference points on the imported photos; reviewing and adjusting the 3D model; component selection with suggestion system and validity checks.

plications into one single platform in terms of technical architecture and user interface.

The purpose of the seven applications is to support the medical technicians with the fabrication, selection and adjustment of prosthesis, orthotics, and mobility solutions. Roughly described the process is: capturing the physiognomy of the patient by different means, selecting components, building the product, and setting up the components for the patient. Depending on the disability of the patient capturing will be done by a combination of the following methods: measuring of dimensions of the patient, photographing the limb, measuring of electrical impulses or scanning of a 3D model.

For a better understanding of the process I describe the ordering of a leg prosthesis: The technician takes specific measurements of the patient and two photographs of the limb. Based on this data the software computes and visualizes a 3D model of the individual socket. The medical technician now reviews and optimizes the result. The 3D data is then sent to the supplier for the fabrication of a thermoplastic test socket. In addition to the socket all necessary components for the prosthesis can be selected and ordered with the software. This selection process is very complex, because it has to fulfill many constraints, such as functionality

of the knee joint and foot, compatibility of different parts and fitting within a certain height. Once the prosthesis is delivered and built together some of the knee joints (the microprocessor controlled ones) have to be configured with the software according to the needs of the patient.

For upper extremity prosthetics myo-electronic prostheses are possible. Those prostheses are electrically powered prostheses driven by electrical impulses transmitted by muscle contractions. The software helps the medical technician measuring the electrical impulses as well as supporting him with the selection of the prosthesis.

At the beginning of the project all seven applications were independent stand alone products. They all had different user interfaces and were build on different technical architectures.

The computer literacy and affinity of the users (the medical technicians) is very diverse. They range from people who hardly ever use the computer – to experts, who adjust every single parameter of the prosthesis to get the maximum comfort for the patient. We had to find a solution which set the entry level low but also allowed maximum control for the experts. The software was planned to be shipped worldwide, hence language and cultural distinctions had to be addressed.

The project was carried out in three locations: head quarter and product management in Germany, software development in London and Vienna, and interaction design in Vienna. Additionally to the already complex nature of the project, this separation of the teams (with their different viewpoints in terms of software architecture and product liability) additionally increased complexity.

The project followed a plan-driven process, and we were involved over a time frame of 12 months. It took the in-house development another 18 months to develop the product. Sadly we were not able to convince the client that we should also be involved in the quality assurance throughout the whole project life cycle. The interaction design team consisted of six team members.

7.2 Audi website

In September 2000 Razorfish Germany was commissioned by Audi, the German car manufacturer, with the redesign of their global website, their national website for Germany and with a concept for the integration of all the dealer websites. Audi was faced with the problem that many small websites were created by different agencies, all of them on different technical platforms. As a result of this, users became confused by differences in the look & feel and the navigation between the sites. Our task was to unite those websites and to provide a system which would allow to update content without HTML skills.

James Kalbach (2002) has also written articles on the Audi website-project, which focus on the information architecture tools used, an adaptive page layout technique and on the usability of right hand versus left hand navigation (see also Kalbach & Bosenick 2006). A description of how the adaptive page layout technique was implemented can be found in Hoffmann (2005).

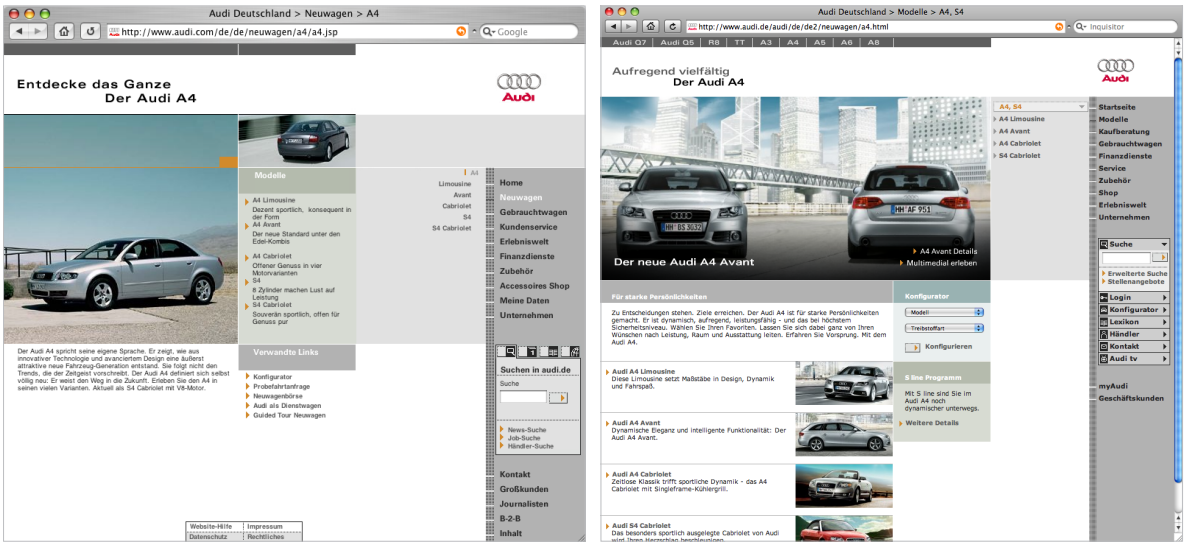


Figure 6: audi.de in 2001 (left) and in 2008 (right).

It was a requirement by the client to follow the *Rational Unified Process* (now IBM Rational Unified Process). At this time not many team members had experience with the process, and therefore engineers and also interaction designers participated in trainings.

The team size ranged from 10 (at the beginning of the project) to 50 members at its peak time. The members were located in three German offices and one office in the USA. Attempts were made to bring the team together into one office, especially during the hot phases of the project. Furthermore many other teams were included within Audi: the marketing and advertising departments, the competence centers responsible for the content, the engineers of the IT department, and the general branding agency of Audi. In the course of the redesign also the web center within Audi was redesigned and rebuilt.

The websites were launched in December 2001.

Rational Unified Process now IBM Rational Unified Process (RUP), *a software engineering process developed and commercialized by the Rational Software Company, captures some of the best practices of the industry. It is use case driven and takes an iterative approach to the software development life cycle* (Leffingwell & Widrig 1999, p 465). See chapters 9.1 (p 33) and 9.3 (p 41) for interaction design in plan-driven processes.

7.3 dpm

Philips Speech Processing commissioned GP designpartners in mid-2005 with the product design and interaction design of their new series of digital voice recorders. The Philips Digital Pocket Memo 9600 series was launched by the end of February 2007.

The Pocket Memo is targeted for the professional market of lawyers and physicians. Our focus was to build an ergonomic and easy-to-use, but also a powerful and stylish product.

The product and interaction design was developed simultaneously and in close collaboration with product management and the engineers at Philips. It was the first time for Philips Speech Processing that they also commissioned the interaction design to an external agency. Especially for the engineers it was a new experience to integrate an external viewpoint,

Figure 7: Digital Pocket Memo 9600 (product photo © Philips Speech Processing).

which was based on user research. Another point was that they were not too eager to change the behavior of the software, because it was already well established in the market and also developed over many years by them and therefore »their baby«.

In this project we followed a typical plan-driven user-centered product design process, with many explorations and iterations in the beginning, and then narrowing it down to get it into production.

7.4 TempRanger

Although many clients of GP designpartners claim that they are following an agile process, in reality most of them pursue a plan-driven process with the eventual iteration. Our project closest to an agile environment was a patent search engine – TempRanger – for Matrixware in 2007. Matrixware applied the *Scrum*-process.

In patent search the toughest goal is not to miss any relevant patent – this could cost a company millions of dollars. Patent researchers are not only looking for a result, but for a needle in a haystack. In addition to a full text search with a sophisticated and complicated query language, TempRanger provided the novel tool of searching for temperatures and

Scrum
is an agile process used to manage and control complex software development projects. By applying iterative, incremental practices it produces a deliverable collection of functionality at the end of every iteration (Schwaber, & Beedle 2001). For a discussion on the integration of design within agile processes see chapter 9.2 (p 35).

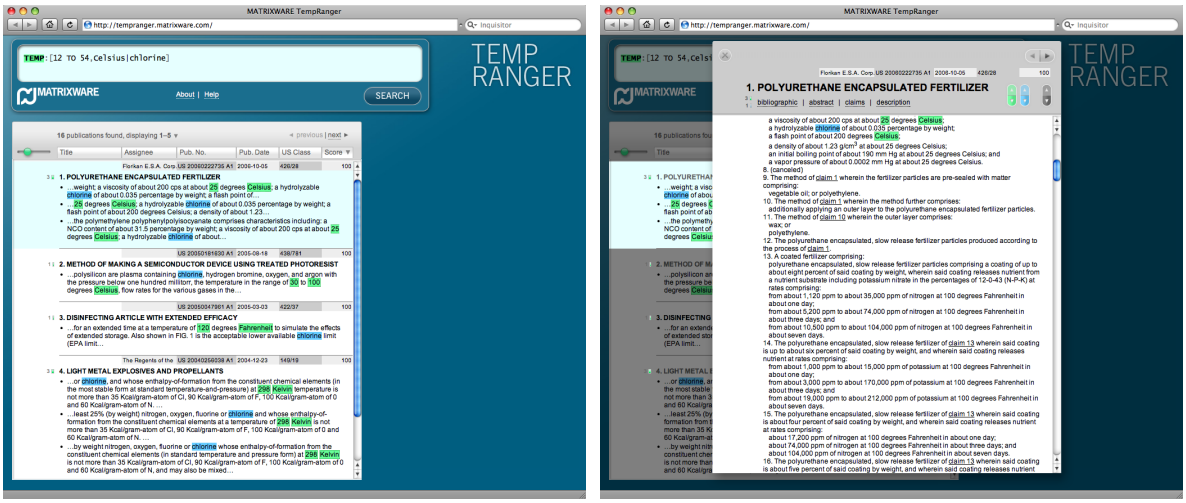


Figure 8: TempRanger search query with result list (left), and with an open document (right).

temperature ranges within documents. At first this sounds as a trivial thing to do, but considering the many different notations of temperatures, e.g. Kelvin vs Celsius, room temperature, boiling point, or 41 vs forty-one vs above forty vs between 39 and 42 degrees, this is a major task. But even if the technology is able to produce those results, it is important to present the results transparent and comprehensible to the patent researchers, because they usually do not trust black boxes (for example sorting by relevance without knowing the ranking algorithm).

The theoretical basis for the temperature search had already been established when we joined the project. The aim of TempRanger was two-fold: showing the process on how Matrixware brings academic research to the industry and the actual temperature search itself in the form of a web application. The project had to be finished within two months to meet the deadline for launching it at a conference. From the very beginning a small team existing of the product manager, designers and both back-end and front-end engineers worked closely together.

8 Setting the stage – conclusions

Purgathofer (2006) argues that with the beginning of the software crisis in the 1960s most of the then available software development processes failed to acknowledge the nature of design work. Consequently a wrong understanding of design and the design process has a large share in developing bad software. Thus tools, methods, and approaches to bring a better understanding of design – design thinking, as described in generation three (p 14) – to software engineering are desperately needed.

Floyd (1992b) states that software development is basically cooperative work, and is characterized by a *dialogical orientation in design* (Floyd 1992b, p 97) among the developers and designers and with the user. Dialogical orientation in design demands sincerity towards the views

of others. Present methods do not respect those different perspectives enough. A dialog orientated design process not only needs to acknowledge the different views, but also to accept the different contributions as interim findings. Those interim findings act as inquiring materials in the overall process of designing software. They are not the final product of design (which is the software itself) but they act as *means for inquiry* (Gedenryd 1998, p 149) for the whole team. Perfect candidates for inquiring materials are boundary objects, shared by both disciplines.

Furthermore, conflicts need to be addressed and not ignored as undesirable disturbances. Trust is essential for such collaborations and can only be build up, if the interests of every individual are factored into subject matter and organizational decisions (Floyd 1992b).

DeGrace & Hulet-Stahl (1990) state that designing software is a wicked problem, and that traditional engineering based software design methods are not a good way to solve them. Moreover, finding processes, methods and tools for software design is a wicked problem in itself – a wicked problem of a wicked problem. Hence finding the right process or method for the development of software is impossible (rule 3: solutions to wicked problems are not true or false, but good or bad). Therefore it is only possible to provide tools to explore and support the team members in framing the given situation and finding the right tools and methods to design software – tools for tinkering with the process and method itself.

Part 3

Teamwork: designers & engineers

In Part 3 I research the collaboration between designers and engineers on software development projects from three perspectives – process, people, and artifacts – on the basis of my experience on many industry projects and case studies from the literature.

Which processes support a designerly approach for software development? How can the ones currently used be modified to integrate design better? Which artifacts act as boundary objects and/or inquiring material in the collaboration between designers and engineers? Who owns user experience? How to evangelize design thinking within in the team?

Those are the questions which will be discussed in the following chapters. I researched the development of software from three perspectives:

- **Process:** Usually some sort of process is followed in a project. Even if no process is followed, then even this is a process: the ad-hoc process. But I will not cover ad-hoc processes, because they are only depending on the participating team members, covered in the following perspective.
- **People:** *No process will substitute for talent and for human relationships* (Kovalchuke 2006). Processes are only a supportive element in projects – the people involved are the key differentiator between good and bad projects.
- **Artifacts:** The third component of projects are artifacts, created on the way to deliver a suitable final product.

9 Process

In software engineering a considerable large range of different process models exists. Those processes are often separated into two major groups: plan-driven methods and agile methods.

Plan-driven methods are based on the assumption that (Boehm 2002):

- Software development is repeatable through controlled processes.
- Requirements of the product can be known and specified prior to development starts.
- Projects are predictable and that they can be planned in detail from start to end.

Examples for plan-driven process models are the waterfall model, the V-Model, and the unified process (Jacobson et al. 1999) and here especially its implementation by IBM Rational, the Rational Unified Process.

Agile methods on the other hand are based on the assumption that desired characteristics and features of the product can only be known if at least a part of it is built. Projects are in general unpredictable and therefore agile methods focus on:

> *Individuals and interactions over processes and tools*
> *Working software over comprehensive documentation*
> *Customer collaboration over contract negotiation*
> *Responding to change over following a plan*
> — Beck et al. 2001

Popular agile models are Scrum, eXtreme Programming (XP), Crystal, Feature Driven Development (FDD), Adaptive Software Development (ASD) and the Dynamic Systems Development Method (DSDM) (Sommerville 2006, p 430 et seq.).

Why even research how design is integrated into plan-driven models? Is not the agile approach the better way to actually design software for the user, with its core element of consistently including the user?

First, from a design point of view not everything is golden within agile processes – many issues remain, such as: Can the users state their needs? Are we not loading off our responsibility to the user? Are we talking to the user of the software or to the customer of the software? Is code the right medium to design?

Secondly, agile methods are not always the best method, each of the models has its strength and weaknesses and its best fit to certain types of projects (Boehm & Turner 2003).

Thirdly, many companies are still using – and possibly will use for a much longer time period – plan-driven models (Mahanti 2006,

Boehm 2006). Many companies apply a mixture of both models. In our experience most of our clients are following plan-driven models. Many of them talk about implementing agile methods, but this takes time. Therefore it is important to find ways to best integrate user-centered design and design thinking approaches into existing process models.

Thus, this chapter discusses the integration of design within those two groups of processes, and available approaches for making user-centered design a more prominent part of software development.

9.1 Design in plan-driven environments

The design phase in most plan-driven software processes is concerned with technical software architecture. The design phase is usually following the requirements phase and is succeeded by the implementation phase. Hence, the design phase on plan-driven methods follows design as problem solving as described in chapter 5.1 (p 13). In recent years the integration of user-centered methods into the software development process improved (Boehm 2006). Nevertheless, this still often means that interaction designers are limited to only styling the product and doing usability tests on the almost finished product (Berkun 2005).

Following are two process models for a better integration of design into the software development process.

The No Silo-model

Following Schön's (1983) point that product development demands attention to problem setting and problem solving Buxton (2007) nicely puts it this way: *getting the design right and the right design*. None is sufficient without the other. If a well designed product gets built the wrong way the final product will be bad. Also a product which gets built perfectly following the specifications (which seemed right at the beginning) will be a failure if the wrong product was specified. Hence Buxton among others (Cooper in Nelson 2002, Purgathofer 2003) demands a distinct design phase at the beginning of the project, in which design is seen as problem setting and solving (cf. chapter 5.3, p 14). Figure 9 shows Buxton's proposal for a simple three phases product life cycle: design, engineering, and sales. But those three phases should not be seen as *silos*: design, engineering, and sales must be involved in all phases. For each phase the focus and the responsible person is stated (Buxton 2007).

The design phase should be understood similar to preproduction in filmmaking. It includes not only the design of the product itself, but also the design of the engineering process and the design of the business model. Continuing this analogy to filmmaking, Buxton (2007, p 79) provocatively asks: *Who is the equivalent to the director? Do they have*

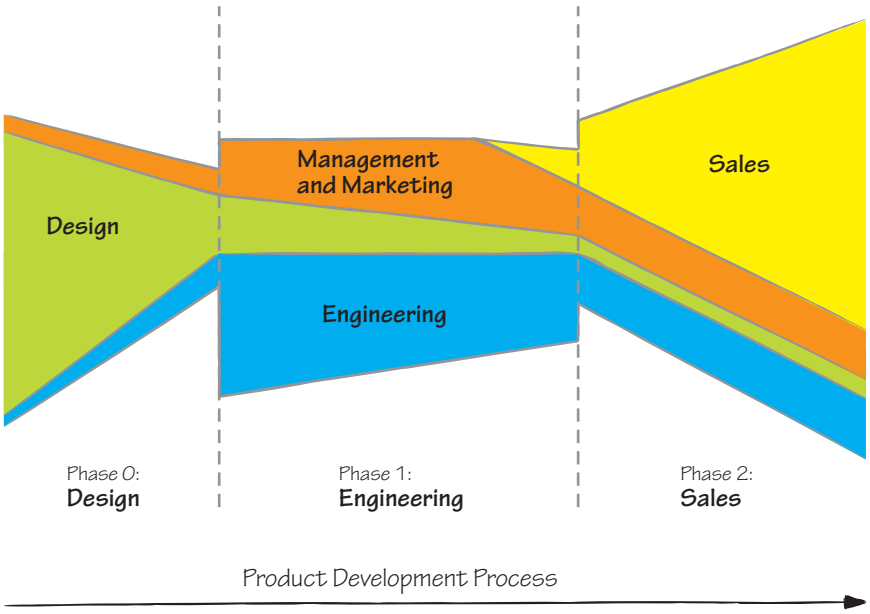


Figure 9: The No Silo-model, product development process based on filmmaking (Buxton 2007, p 76).

comparable power and responsibility? If not, why not? If not, why do we believe that the integrity of the design will be maintained?. The same questions should also be asked for engineering and business.

The butterfly model

Purgathofer (2003) presents a similar model, in which design and implementation is clearly divided into two phases (figure 10). He states that in the design phase the engineers should be included and seen as *user* of the design – because for implementing it, they are using the design. Therefore they have a similar stake in product design as the actual users. The engineers can give input to the design – but as with real users they give only suggestions and it is the designers right to factor them in or not. Once the design is handed over to the engineers the designers act as *clients* in the project.

The engineers usually have other interests in the design than the users. I therefore would also call the engineers *clients* rather than *users* during the design phase. Similar to the client – who for enterprise level solutions often also is the customer to pay for the product, but seldom the user they have to be convinced of the design. The design has to be *sold* to them (cf. chapter 10.4, p 55). At a workshop the engineers swap their initial role of being the client with the designers.

This approach has much in common with the building of a house. The architect designs the house and the structural engineer sets it up. In the building phase the architect acts as the spokesman for the future residents of the house.

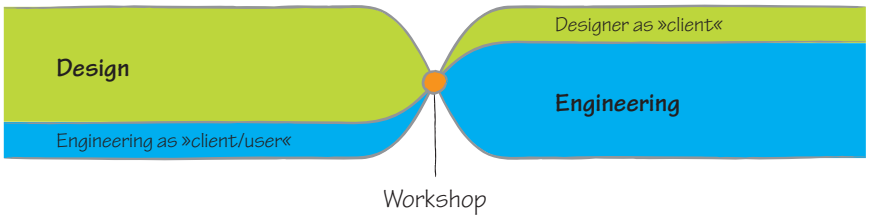


Figure 10: The butterfly model (adapted with the engineers as users and *clients* of design) (Purgathofer 2003, p 321).

9.2 Design in agile environments

As with plan-driven methods also with agile methods the role of the designers is not clearly defined. A major part within the agile methods is the direct involvement of the user in every iteration. The absence of design involvement – and on the other hand the unreflecting direct involvement of the users – are the main issues in the available research on design within in agile methods.

Interaction design vs extreme programming

An interview of Alan Cooper and Kent Beck in Nelson (2002) discusses the collaboration between interaction designers and engineers within extreme programming environments. Cooper argues that *neither users nor customers can articulate what it is they want, nor can they evaluate it when they see it.* Besides that, in my experience it is often very hard to get direct access to the user, and if it is possible only limited time is available. So the engineers usually get the unfiltered view of the customers of the software, who rarely are the users of the software. Consequently the engineers do not receive the needs of the real users, but an interpreted version of what the customers think their users need from the software. Cooper further argues that software has two sides, the fast and error-free side regarding the hardware, and the slow and error-prone side which faces the human (cf. *internal* and *external design*, p 9). Furthermore that engineers do not have the right skills to work on both sides. He compares it to architecture where the building owner does not talk to the builder but to the architect, and therefore the programmer should not talk to the user.

Cooper continues that interaction design is closer to requirements engineering than it is to interface design. He concludes in a similar way to Buxton (2007) and Purgathofer (2003) that the design has to be finished before implementation starts: *There's enormous cost in writing code, but the real cost in writing code is that code never dies. If you can think this stuff through before you start pouring the concrete of code, you get significantly better results.* Naturally Beck does not agree on this, because *the interaction designer becomes a bottleneck, because all the decision-making comes to this one central point.* In my opinion this is a resource issue and not a process issue. Because the same is true when the engineers are talking directly with the users, then the users become the bottleneck. Larger

projects need more interaction designers, and those have to interact with all the other departments (see chapter 10.6, p 67).

Both seem to have a diametral view on how products can be built, as Beck's put it: *That's one of the base assumptions that seems to be very different in our thinking – that you can build a program that improves with age instead of having it born powerful and gradually degrading until you just think, »Oh, this is crap and we have to start over«.* An interesting thing happened during the interview. Beck started using the term interaction team instead of the customer team, which is in my view a step in the right direction. Cooper neither believes in extreme programming nor that interaction design can be integrated into the former. He always comes back to a plan-driven approach and follows a very similar line to the one described above in the butterfly model: *During the design phase, the interaction designer works closely with the customers. During the detailed design phase, the interaction designer works closely with the programmers. There's a crossover point in the beginning of the design phase where the programmers work for the designer. Then, at a certain point the leadership changes so that now the designers work for the implementers. You could call these »phases« – I don't – but it's working together.* On the other hand Beck is much more open to not directly involving the user into the process, and instead use interaction designers as a translator or filter between users and engineers. He nicely concludes: *To me, the shining city on the hill is to create a process that uses XP engineering and the story writing out of interaction design. This could create something that's really far more effective than either of those two things in isolation.* I believe this is the only way to go within agile environments. But this does not need to mean that agile environments are always the better model for any project.

Up-front design

In a field study Chamberlain et al. (2006) researched the integration of user-centered design and agile development within three teams. Both methods showed similarities, but also differences in their approaches. The main similarities are:

- Both rely on iterative processes and build on findings gathered in the previous circle.
- Both are focusing on the early and constant involvement of the user (within agile environments this is often the customer).
- Both stress the importance of team collaboration and communication.

The differences can be found in:

- Who is the boss? In agile approaches the customer is the boss, he pays the bill. In UCD approaches the user is the boss, despite the fact that still the customer pays the bill.
- Agile approaches strive for minimal documentation. UCD asks for certain design documents to improve communication.
- UCD seeks to understand the users fully at the very beginning before any implementation is done. Agile development is against up-front design at the cost of coding.

All investigated teams had already experience in combining UCD and agile approaches in the past, but all used different kinds of integration of the two. For the agile approach the teams used either Scrum or extreme programming.

Chamberlain et al. (2006) discovered that all three teams were applying some degree of up-front design, as did Ferreira et al. (2007) in her qualitative study, before starting the iterative cycle of the agile method. All teams were convinced that up-front interaction design has advantages. The benefits by doing up-front design are that the designers are only marginally limited by the technology used, and therefore were faster and could factor in user feedback earlier. Doing up-front design also had positive effects on budget, task prioritization and usability. The interviewed teams were doing about 70–90% design work up-front. The remaining part was done during implementation, giving the interaction designers more insight on how the actual software is working. Also users could reflect better on real working software compared to prototypes.

The study of Ferreira et al. (2007) also discovered an interesting fact about the engineers' view on up-front design. In the participants' view agile development warns against up-front code design – but not against up-front interaction design. Ferreira et al. (2007) concludes that this up-front work might be seen by engineers as analysis and not as design: *Agile development advocates are wary of up-front design because it represents premature commitment, but if it is analysis and does not involve commitment, then it does not constitute the same kind of danger.*

So, is agile development just for the engineering phase? The study shows that in practice the teams are using a combination of a plan-driven and agile method, a two-phase-model similar to the butterfly model. With a user-centered approach in the first phase and an agile approach in the second phase.

Despite some problems, user-centered design and agile methods are compatible. The problems include power aspects between engineers and designers; different times to produce results for each iteration (design is

usually faster than development); lack of communication if the members are not included in all phases of the project; an unwillingness to understand the different aspects and needs of each element of the project; a different understanding of how much the user should be contributing to the project. To overcome these problems Chamberlain et al. (2006) came up with five principles to integrate UCD and agile methods:

1. ***User Involvement*** *– the user should be involved in the development process but also supported by a number of other roles within the team, such as having a proxy user on the team.*
2. ***Collaboration and Culture*** *– the designers and developers must be willing to communicate and work together extremely closely, on a day to day basis. Likewise the customer should also be an active member of the team not just a passive bystander.*
3. ***Prototyping*** *– the designers must be willing to »feed the developers« with prototypes and user feedback on a cycle that works for everyone involved.*
4. ***Project Life Cycle*** *– UCD practitioners must be given ample time in order to discover the basic needs of their users before any code gets released into the shared coding environment.*
5. ***Project Management*** *– Finally, the agile/UCD integration must exist within a cohesive project management framework that facilitates without being overly bureaucratic or prescriptive.*

— Chamberlain et al. 2006

Integrated iterative interaction design

Miller (2005) reports from a successful integration of interaction design and development within agile environments at Alias, a leading provider of 3D software. Before applying agile methods they were following a standard waterfall process. Despite a large and good integrated usability and interaction design team they ran into problems. Often the designers did not have enough time to specify all the features needed, because the engineers started implementing different features in parallel. Much was based on guess work, without verification with actual users. To avoid this situation, designers often started early, even before the end of the requirements-phase, to have the designs ready once needed. But this also meant a waste of resources for features which would not be implemented.

Already by applying agile methods, with its focus on only a few features and on producing a working software in every iteration, the wasting of resources could be avoided. Development changed from parallel work on many features to a team effort on fewer features.

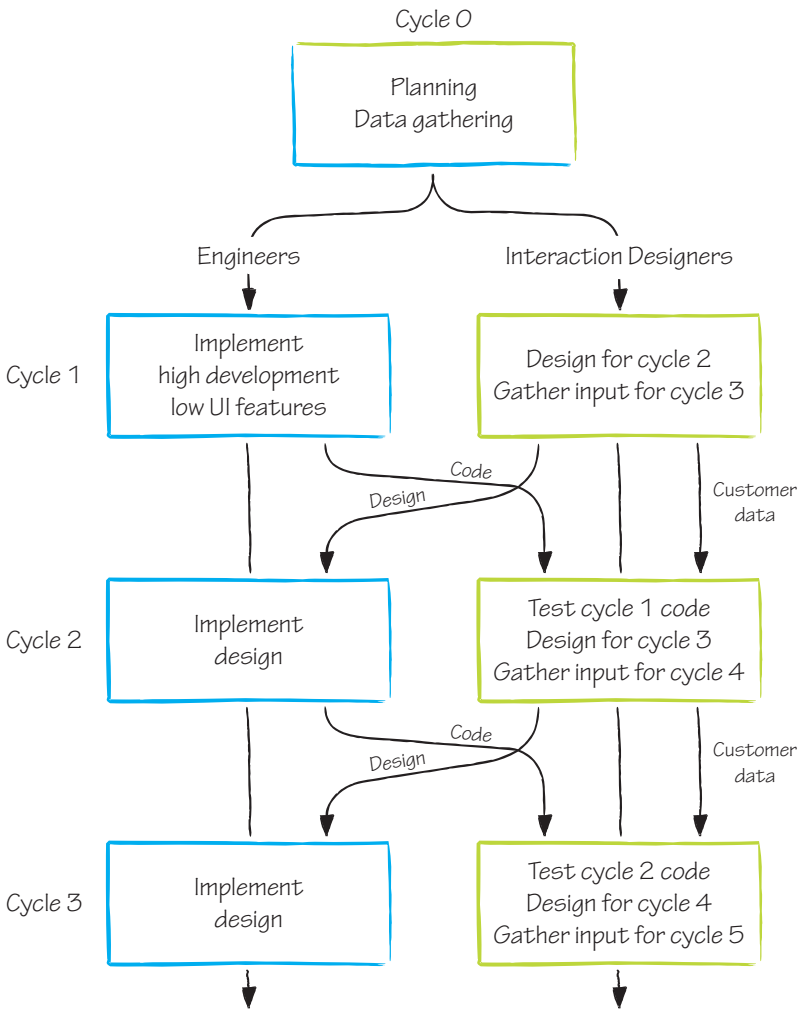


Figure 11: The dual tracks within a parallel iterative development (Miller 2005).

At Alias they already used a large repertoire of user research methods for a long time and all had the skills to apply them. The challenge of integrating them into an agile environment remained *the timing of these activities* (Miller 2005).

Figure 11 shows their adoption of the process for integrating interaction design and development within agile environments. In cycle 0 user input was gathered, the needs identified and then product manager, development lead and interaction design lead jointly prioritized the product features.

Cycle 1 differs from the following cycles, because the designers did not have time to prepare the designs for this iteration, as they only just got the feature list. Thus, development started implementing features with little design but high development effort, such as saving files in different formats. The interaction designers started with designing features for cycle 2. They also built small isolated prototypes and conducted usability tests with them. At the same occasion they furthermore gathered input for the features to be designed in cycle 2 and implemented in cycle 3.

Beginning with cycle 2 the wheel started turning. The developers started implementing the features, which were designed in cycle 1 by the designers. The designers conducted usability tests with the ever growing application from the previous cycle (in this case cycle 1). This had the advantage that it already included those changes which had to be applied because of technical implementation changes and it furthermore showed how features interacted with one another. Additionally they designed and built prototypes for cycle 3 and gathered user input for cycle 4.

But the communication between the teams was not limited to *just throwing the designs over the wall and the code back* (Miller 2005). In daily Scrum meetings the developers could follow the design activities. If needed additional interface presentations took place to gather input from the engineers regarding the technical feasibility of certain designs for the next cycle. On the other hand designers answered questions and solved problems which arouse during the implementation.

By applying this parallel-track process Miller (2005) pointed out large wins for both the interaction designers and the developers. The wins for the interaction designers are:

- Not wasting time for designing features which were not used.
- Saving time by combining usability tests and contextual inquiry within the same trip to the users.
- Getting timely feedback not only from the users but also from the market, e.g. a new release of a competing software.

The developers had the following wins:

- Maximizing coding time, because they did not need to wait for the designers to complete the prototypes and usability tests.
- No waste of effort for implementing multiple alternative design concepts for innovative interface parts.

To be fair it has to be said in reality they could not always follow the process as shown in figure 11, because some of the features needed more iterations to fulfill all of the design goals. Overall the designers sticked to the general rule: *the design is timed so that it is ready just as the cycle that it is needed in starts* (Miller 2005). Miller (2005) concludes that *they were able to maximize the quantity and impact of customer input by having the interaction designers work in a parallel and highly connected track alongside of the developers.*

9.3 Our experience with processes

About 80% of our projects are carried out within plan-driven environments. Although many of our clients explicitly state that they are using – or plan to use – agile processes, in reality this is not the fact. Another fact we experienced is that many of our clients have very detailed processes with specific deliverables – in theory. In practice those processes are only loosely followed, if at all. Before I describe two cases, one within a plan-driven environment and the other within an agile environment, I will briefly describe our own process.

We are following a user-centered design process:

- **Explore:** To fully understand the situation and needs of the users and clients we perform interviews and observations. We start with stakeholder interviews and workshops within the clients company to understand expectations and goals for the product. We then interview and observe users, develop personas and usage scenarios. In parallel we analyze the current solution and future developments. Based on the findings we come up with initial concepts.
- **Shape:** That is our most creative and iterative phase. We do brainstorming-sessions to find an overall metaphor and generator concept. We build early low-fidelity and semi-functional prototypes, test them and start over again. This process allows us to test new things but with the safety to stop wrong directions early. Most of the problems will be discovered in early stages, with not much time and effort invested.
- **Complete:** Although documentation takes place all the time in parallel, in this phase it is time to prepare our final deliverables in full detail and in an implementation friendly way. The user interface style guide specifies guiding principles for the used interaction elements, their usage and arrangement, as well as the general look & feel. The functional specifications define the functionality and interaction concepts – based on the general rules of the style guide. In addition graphical elements and icons will be developed and prepared ready for implementation.
- **Standby:** During the implementation phase design reviews and workshops assure the correct implementation based on the concept.

Another phenomenon we often experience is that we are asked to produce a style guide or to design the user interface of an already existing application for which currently the development of the next version is in progress. This happened for instance in the project fuse. Here we were asked to develop a user interface style guide to combine the clients range of seven independent applications into one single platform and to help

the distributed development team to follow the same user interface style. But to be able to produce not just a shallow cover for making the applications only pretty we had to gain a better understanding of the context. In the process of convincing our client about this fact, we figured out that – despite otherwise stated – the requirements were at best only roughly covered.

Plan-driven environments

On the project fuse, our client did not follow their stated formal plan-driven process, they clearly worked in a plan-driven way – more or less a traditional waterfall approach. Hence we followed our UCD process stated above and tried to include the engineers as much as possible. Not only because they were key knowledge carriers, but also to get them into a user-centered thinking and away from their technical driven approach. At the beginning we were not taken seriously, for instance, with personas, but over the course of the project even the engineers started using them (cf. chapter 11.1, p 73). Through scenarios (and prototypes later on) we were able to clarify our understanding of the situation and to transport our concepts. Additionally those artifacts helped product managers, sales people, engineers and users to reflect on their input. The requirements we got at the start of the project were hereby not only questioned, but altered and extended and new ones were discovered. Over the course of a few iterations the requirements got more and more stable and we got a better feeling for what is important and has to be included in version 1 – what can be postponed for a later release.

In the beginning of the project we were concerned about getting in conflict with the engineers, because of taking over a part they were responsible before we came onto the project. But after gaining the trust of the engineers – especially through our prototypes and our ability to understand their technical concerns – they gave us the time and freedom to explore and detail the concepts. In the meantime the engineers worked on technical features with little impact on the user interface and high development costs, such as the photo processing and 3D model calculations. In addition they synchronized the code base of the different development teams and set up a framework for fuse. Moreover they checked the technical feasibility of our concepts.

At the very beginning of the project they provided us with some documents, each a few pages long, and called them requirements specifications. Those documents included only some of the features, and described them in different levels of detail. For example, one of the documents was the rough description of their planned framework within which all different applications should work. The other documents were documents written by the development teams individually to cover the applications

they had already developed. Our final deliverable on fuse were functional specifications (p 106), with detailed descriptions and flows (p 107) about the behavior of the software. We furthermore delivered detailed and pixel accurate screen designs (p 110). Our functional specifications then formed the main part of the requirement specifications, which were used for developing the application.

The biggest challenge in our projects is to convince the client that we should also be involved in the quality assurance process after the design phase as shown in Buxton's no silo-model (figure 9, p 34). In our process we call it standby-phase, and it is planned to last over the whole product development life cycle, and ideally fixes minor problems for this version. Additionally in this phase we gather features and issues for the next iteration of the product. At the beginning of the project our clients see the standby-phase as very valuable, but in reality we are only rarely involved during the whole product life cycle. The clients are usually more than happy with the depth and quality of our functional specifications – despite our disclaimer that we are only able to cover about 80 to 90% of the project definition within functional specifications. Changing requirements together with the discovery of technical issues during the implementation phase makes it impossible to cover the missing 10 to 20%. On fuse we got some questions about minor issues at the beginning of the implementation phase, but almost none after that.

On the Audi-project the situation was quite different. Here at least one information architect was involved throughout the whole product life cycle, until the website went online. After the design phase the job of the information architects – together with the quality assurance people – was to assure that the developed product meets the specified functionality and look & feel. We entered our issues directly in the bug tracking system. This assured that the issues we reported were taken care off. In the hot phase – shortly before the launch – the bug tracking tool was the only tool everyone takes notice of, no one would look at reports regarding design issues.

One of the reasons for the involvement of design in later phases was that the complete product was developed by one company, as the client had commissioned the complete website – compared to fuse, on which we were only commissioned with the design phase.

This is different in agile environments. On TempRanger, which I will describe below, we were involved right to the end: until the project was shipped.

Agile environments

On TempRanger the development team followed a Scrum process. The project was set up nicely with all the project management documents in place, and everyone on the team had a clear vision of what they wanted to accomplish – from a technological point of view. But they had only a vague picture – at best – of how the experience should be for the user. When we were brought on the project some of the back end temperature extracting routines had already been developed. As our relationship with the client was planned as a long lasting one, we asked them for a very short explore phase to gain a broader understanding of the business and processes in patent research and the stakeholders and users involved. The goal of this phase was to come up with rough concepts for patent searching in general and also for searching for temperature specifically. Additionally a rough plan laying out five iterations to implement some of them within the short project time frame was delivered (compare to findings of Chamberlain et al. (2006) and Ferreira et al. (2007)). We convinced them to not further work on the actual search interface, but to stay focused on the indexing routines for the database, so that we would not be completely constrained just by technical issues.

At the end of the explore phase we presented the concepts in a meeting with the project team. Furthermore we proposed how to build TempRanger in five iterations, with a functional and complete product in every iteration. Already at this time it was clear that we were not able to implement all five iterations, but that we would try nevertheless to get as far as possible during the short time left.

From that point on we worked very closely with the front-end developers, detailed the concepts and prepared the graphical elements (this would comply with a combination of our shape and complete phase in our plan-driven UCD process). Although this was done a little bit ahead of the engineers, in my opinion the project ran more chaotic as described in Miller (2005). We did not strictly follow the iterations. It was more or less an ever growing product, over probably far more, but shorter iterations than planned. Especially at the end the designers and front end developers were sitting together to directly change the code and see the result and decide ad-hoc how some features should behave. Also some of the extraction functionality did not work out as planned, thus we had to change some of the behavior of the search result representation. Because we were involved until the last day before launch, decisions were not only driven by technical feasibility, but also with the user in mind. It was not for instance possible to provide all statistics of the different search result as planned (only temperature results, full text results, and both of them together). The engineers still wanted to provide some of the statistics, to show some sophisticated numbers. But this did not make sense for the

overall product and would have confused the user more than helped. With being still involved in the project we were able to change the concepts and find a different and still understandable form for the search result statistics.

Feature-wise we did not reach our planned goal, but we were able to deliver a product which showed how Matrixware approaches patent searching and the temperature extraction capability. The application went live, as planned, at the conference.

Coming back to the beginning of the project: With an early explore phase we also had an additional intention in mind: To transfer our understanding and approach of design. That it is not just the graphical user interface with its nice icons, but a holistic understanding of the domain, user and context. As we experienced on the follow-up project (which we are currently working on), we did not fully succeed with this, because the engineers still call us user interface designers and still claim that we are responsible for the usability only. In their mind-set the features and functionality is still driven by the product owner and communicated to the engineers. Only thereafter also the designers were planned to be involved. This is currently changing. We were able to slowly convince them that we have to be included much earlier – for example by being included in working out the user stories (cf. *Scenarios and user stories in the follow-up project of TempRanger*, p 85) together with the product owner and engineers and factor in results from ongoing user research.

9.4 Summary

In practice usually not only one single process is followed rigidly. Elements of both plan-driven and agile methods are combined. In fact modern processes, such as the IBM Rational Unified Process, are a compilation of best practices of several other proven software engineering concepts (Sommerville 2006, p 112). Iterative and incremental development is combined with object oriented design using UML and use case driven development. As shown in the studies of Chamberlain et al. (2006) and Miller (2005) and as experienced in our projects also agile processes are using elements of plan-driven processes in the real world.

Some of the processes described above are very detailed, down to specific deliverables (e.g. RUP), others are very high level and describe a rather general approach (e.g. the butterfly model). We experienced that the single most important thing is the understanding of the roles and their according disciplines within a team – irrespective of how clear a process is defined. Only an understanding of needs and approaches of the other team members fosters the respect among all people involved. And only then it is possible to deliver a successful project.

Regardless which process a team or company follows it is important to note that the process always needs to be adjusted:

> *If your engineers aren't arguing about the way they develop software* ALL THE TIME, *they're becoming stagnant and that trickles down to your pitch and trickles up to your product.*
> — Lopp 2007, p 78

For Scott Berkun a good process is one process which includes a mechanism to constantly review itself and to invoke change if necessary:

> *Good processes include a process for changing or eliminating the process. Because projects and teams are changing all the time, a process that is useful or necessary one month may not be useful or necessary the next month. The process itself must have a built-in mechanism for deciding when it's no longer useful or when it needs to be updated to make it useful again. Never assume that a process will go on forever, and avoid defining jobs around processes for this reason. […] make people responsible for the effects and results the processes have on the project.*
> — Berkun 2005, p 189

Therefore I do not believe in any process per se. It is not about the process, but about how people implement it. I am confident a successful product can only be delivered by a motivated team which has an understanding of the disciplines involved.

Most research on design in software projects talk about new processes or changing existing processes. Ways have to be found to integrate design into existing process, e.g. into the waterfall- or V-model, without changing them too much. This might be frustrating for the designers, but over time and with an increasing number of successful projects their role will get more and more important. If they are prepared to assimilate within an existing process they can use this to their own advantage as well as to the products.

In the following chapters I investigate different aspects of how this mutual understanding can be fostered and I suggest tools to encourage the team members to talk about the process itself and to gain an understanding of the different disciplines and the approaches involved.

> *No process will substitute for talent and for human relationships. […] Process is not solution, it is only facilitator – it can remove some of the hindrances to personal contribution and to effective team work, but it will not replace talent. Emphasize*

> *outcomes, play to personal strengths and process will evolve around available personalities.* — Kovalchuke 2006

No process will substitute for talent and for human relationships. – A perfect bridge to the next chapter: *People*.

10 People

Who owns user experience? What skills are needed to influence a team so that it is able to deliver a good user experience to the user? Is a catalyst role needed in a project? Those are the questions which drive this chapter.

10.1 Who owns user experience?

interactions (2005) dedicated a whole issue to the discourse about who owns user experience (UX). The discussion of the ownership of user experience is twofold:

1. Who owns UX within the design and human computer interaction community?
2. Who owns UX in regard to a whole project team or even at a company level?

Can user experience be designed at all?
For an interesting, although more philosophical discussion whether user experience can be owned by anyone but the user see Porter (2007) and (Sherman 2007). They state that the experience belongs to the user and that one can only create artifacts and conditions to influence this experience. Despite being an interesting thought, I will still use the term *designing the user experience* as it is used in the community: that is for designing the artifacts which influence the user experience.

The common conclusion to the first question is to stop the battle and discussion among the different specialized disciplines (such as interaction design, information architecture, interface design, user research, and usability), and to pull together on one string (Anderson et al. 2005, Bogaards & Priester 2005, Gabriel-Petit 2005, Goodman 2005, Sherman & Quesenbery 2005, Tognazzini 2005). Gabriel-Petit (2005) calls for collaboration instead of competition:

> *In a culture of collaboration rather than competition, there is a healthy cross-pollination of ideas among all the professional disciplines.* — Gabriel-Petit 2005

The debate to the second question is more diverse. The viewpoints can be grouped into three categories:

› **The business people:** Bogaards & Priester (2005) and Knemeyer (2005) argue that user experience is owned by those who make the business decisions.

  - *They know that, in the end, UX is about the creation of great products that people really want and can enjoy using.*
  - *They understand the importance of delivering a high-quality user experience in this rapidly »virtualizing« world.*

> - *Only they have the power to structure their organizations in such a way that every employee contributes to the delivery of a high-quality user experience.*
> — Bogaards & Priester 2005

Interaction designers are still not in the position to own user experience. They are depended on the vision of the CEOs and even on the human resource people, who decide on new employees (Knemeyer 2005). For Norman (2005) management is only responsible, when it comes to conflicts, otherwise the whole team is in charge.

- **The engineers:** Tognazzini's argument goes in a similar direction, that the value of interaction designers still is not recognized, but his conclusion is a different one. For him the engineers own user experience, because only few companies employ designers, and therefore the design is left to the engineers (Tognazzini 2005).
- **The team:** The widespread understanding is that no single discipline owns user experience (Anderson et al. 2005, Arnowitz & Dykstra-Erickson 2005a, Ashley & Desmond 2005, Gabriel-Petit 2005, Norman 2005, Vanka 2007). For Anderson et al. (2005) *the best user experience is the product of many different disciplines working together* and that the *focus is collaboration, not ownership*. Arnowitz & Dykstra-Erickson (2005a) state that *user experience design is by its nature collaborative*.

Interestingly no one argues that the interaction designers own user experience.

Everyone in a project team is responsible for something: product managers own product vision and strategy; project managers care about delivering the project on time and within budget; engineers are responsible for the internal design of the product, code development and the quality of the code; and marketeers are the owners of marketing communications and product campaigns. Therefore, for Arnowitz & Dykstra-Erickson (2005a) it is only logical that interaction designers take responsibility for user experience issues.

Gabriel-Petit (2005) and Hawdale (2005) argue that *a leader with a vision* (Hawdale 2005) contributes most to a successful user experience. This person can be from every involved discipline, be it product management, engineering or user experience.

In my opinion user experience is owned by the whole team and anyone can and should take the lead. But in my experience this most often is an interaction designer.

I believe that a good user experience can only be delivered if everyone of the team contributes his share. Hence everyone is responsible for it. But this often means that in the end no one cares about the user experience, because it was in everyone's responsibility, and in reality this means no one takes this responsibility. I believe that someone has to take the lead and responsibility for the user experience itself and to bring the awareness for user experience into the team. This person can be from any discipline, but has to be a design thinking person. In my experience of many projects those people were most often (interaction) designers.

10.2 Design as a role: Interaction designers

Are we all designers as Norman (2004b, epilogue) states? Buxton comments sarcastically: Is everyone then also a mathematician? Because everyone counts his own change. Just because everyone selects his own furniture or clothes does not make him a designer. Thus, no, not everyone is a designer (Buxton 2007, pp 95–97).

The discussion above shows how broadly the term *designer* is used. In the design community, designer refers to someone who has practical experience or training in disciplines such as product design, graphic design, or interaction design. Buxton summarizes that one important characteristic for designers is that they sketch (Buxton 2007, pp 95–97). In the next chapter capabilities of an interaction designer are discussed in more detail.

In projects many different job titles might be used for those members responsible for the user experience: information architect, interaction designer, graphic designer, user interface engineer, usability engineer, etc. Based on a long and interesting discussion on the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007) and Tognazzini's call:

> *Use the title »Interaction Designer«. It may not be perfect, but what's important is that potential employers hear and see a single, unified term.* — Tognazzini 2005

I use the term *interaction designer* as an umbrella for all (design) disciplines concerned about the user perspective (figure 12).

In projects the interaction designers are typically seen as the *designers*, and they usually have the best knowledge about design methods of the third generation (p 14) . Or as Vanka (2007) puts it: *The interaction designers are responsible for raising the design IQ within the team.*

What capabilities are needed to support the team in delivering a product with a good user experience?

Role
is defined by *Merriam-Webster's Online Dictionary* (2008) as follows:
(1): a character assigned or assumed <had to take on the role of both father and mother>
(2): a socially expected behavior pattern usually determined by an individual's status in a particular society

I use the term *role* defined in the above way. One can take on more roles in a project. Furthermore, the same person can act in different roles in different projects. Of course, the person acting in a specific role is influenced by his specific knowledge and skill set, but his responsibilities lay in his defined role on a certain project. I am using role instead of discipline or job title (1) to break up the one to one relationship between a person and his job title, (2) to tie the responsibilities of a role to the according person, compared to disciplines which are too broad and general and rarely talk about responsibilities. A role in a project is responsible for certain tasks and deliverables. On small project teams one person will most likely fulfill more roles, compared to large teams, which are often staffed with specialists focusing on one or few roles.

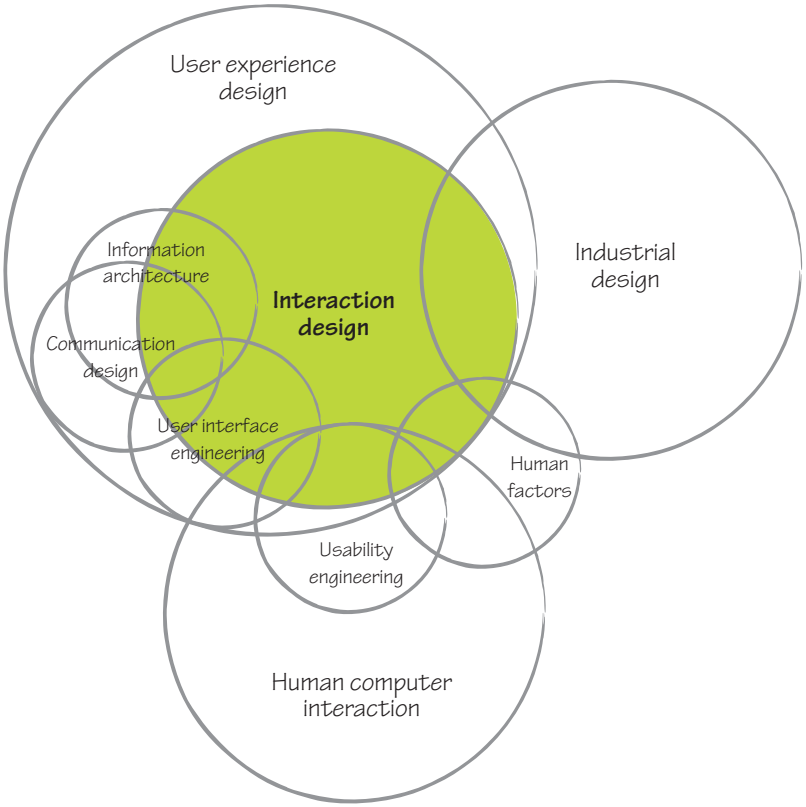


Figure 12: The relationship and overlapping of the different disciplines regarding the user perspective (Saffer 2006, p 17).

10.3 Capabilities of an interaction designer

Interaction design is still a very young discipline. Many discussions about its naming and responsibilities take place within the community. One of this ongoing discussions covers the capabilities an interaction designer should have. What does the community think is the role of an interaction designer, what are the skills he should have?

Moggridge defines five design skills a designer should have as follows:

> › *To synthesize a solution from all of the relevant constraints, understanding everything that will make a difference to the result*
> › *To frame, or reframe, the problem and objective*
> › *To create and envision alternatives*
> › *To select from those alternatives, knowing intuitively how to choose the best approach*
> › *To visualize and prototype the intended solution*
> — Moggridge 2006, p 649

Buxton goes in a similar direction, but emphasizes the activity of sketching:

> *To me, a designer is someone who integrates these points into their thinking and working life:*
> › *The nature of design*
> › *The importance of sketching*
> › *The plurality and variation of approaches that sketching affords*
> › *The capacity for fluid, simple, rich, and spontaneous annotation*
> › *The social nature of design, in terms of both people and sketches*
> › *The importance of having persistent displays that enable work to »bake in« to the shared consciousness*
> › *The fundamental role of critique in design education and process*
> — Buxton 2007, p 201

Robert Reimann (at that time president of the Interaction Design Association) separates the capabilities into core skills and additional skills, such as business and communication skills, which are more or less important for any other role as well. The core skills are:

> › *Research techniques (online and paper-based)*
> › *Ethnography and discovery (studying user goals, motivations, work patterns)*
> › *User modeling (persona and scenario creation; role-playing)*
> › *Product design (product-level interaction principles and concepts)*
> › *Interaction design (function-level interaction principles and concepts)*
> › *Interface design (GUI component-level interaction principles and concepts)*
> › *Information architecture/design (content structure/presentation principles and concepts)*
> — Robert Reimann in the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007)

For Rosenberg prototyping is a key skill of interaction designers:

> *Building prototypes remains the lingua franca of all design professions.* — Rosenberg 2006

The positions on how much technical understanding an interaction designer should have are more diverse. In a discussion on the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007) the range is from no understanding at all to actual coding/programming experience. But the majority in the discussion and others state that a certain understanding of the medium one is designing for is crucial, to understand the constraints and possibilities and to be able to communicate with your team members (Cooper 2004, p 233; David Malouf in the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007); Purgathofer 2003, p 321; Jared Spool in the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007); Winograd et al. 1996, p 9 + p 297).

> *[…] knowing a little code is helpful. I disagree, however, that all interaction designers need to be coders. What's necessary is:*
> - *an ability to understand the constraints and possibilities of the medium you are working in, so that you can best utilize (and work around) them*
> - *and an ability to effectively communicate to those who have to make your design work*
>
> *I don't think you have to know code (and materials, for those of us working with physical devices) to do either of those things... but it helps.*
> — Dan Saffer in the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007)

In my experience being able to gain domain knowledge quickly is the key capability a interaction designer should have. He not only needs to immerse himself into the role of the user but also needs to be taken serious by other team members, as many of our projects show. The capability of gaining domain knowledge quickly often gets emphasized in the testimonials of our clients.

Lash & Baum are calling for more initiative of interaction designers:

- *Lead! – Don't wait for a product manager or other team member to request a specific research activity or design deliverable.*
- *Ask product managers what their goals are for their product.*
- *Don't just propose designs and wait for a decision. Prepare yourself to discuss the impacts of specific design choices. Present the reasons behind your decisions and how they relate back to the broader vision and objectives.*
- *Make strong recommendations and include the big picture.*

› *Help product managers get out of the office! – Invite them along next time, and if they say no, keep asking.*
— Lash & Baum 2007b

To top the capabilities chapter off I would like to add some inspirational qualities an interaction designer should have:

A great interaction designer is someone who:
› *is fanatical about making the world a better place*
› *can synthesize understanding and creativity to produce magical insight*
› *can inspire others to share their vision*
› *whose leadership infects others so much that together they create amazing things*
› *can extrude opportunity where others only see status-quo*
› *creates things that engage and amaze*
› *creates things that forces people to shift their expectations and world view*
› *who remains true to their vision and authentic to themselves*
— Gabriel White in the *IxDA-mailing list: What sets the »best« interaction designers apart?* (2007)

10.4 Getting your point across

Getting your point across is important, obvious and applies to every discipline, but I would like to focus here especially on designers getting their point across to the other disciplines. The job of designers is to get the subject of a digital product across to the users and create a perfect user experience, but they often lack the capabilities to get the point across to the other team members, and herewith risk that they only have little influence in creating the user experience delivered with the product. Often the planned user experience only reaches the concept stage. The engineers are the ones who have the ultimate handle on the final product and according to Tognazzini (2005) they own the user experience. Therefore it is important that designers are not only good in convincing the client of the concepts, but also that they become experts in transporting the concept of the overall product and the design process to the engineers.

Selling the concepts – making the design rationale visible

Design is 50% selling is a common saying within the design community. The presentation of the concept to the client is of similar importance as the act of designing itself. Usually it is not enough to just show the

design artifact, it has to be *sold* to the client, so that he buys into the ideas, metaphors and ideas.

To convince the recipients of the concepts designers have to a*rgue for their outcomes and their value* (Stolterman 2008). Designers do this by making their judgements and their design rationale visible. The design rationale describes the drivers, motivations and reasons that take place during the design process:

> 1. *An expression of the relationships between a designed artifact, its purpose, the designer's conceptualization, and the contextual constraints on realizing the purpose.*
> 2. *The logical reasons given to justify a designed artifact.*
> 3. *A notation for the logical reasons for a designed artifact.*
> 4. *A method of designing an artifact whereby the reasons for it are made explicit.*
> 5. *Documentation of (a) the reasons for the design of an artifact, (b) the stages or steps of the design process, (c) the history of the design and its context.*
> 6. *An explanation of why a designed artifact (or some feature of an artifact) is the way it is.*
>
> — Moran & Carroll 1996 in Louridas & Loucopoulos 2000

Horner & Atwood (2006) argue that the goal of design rationale research is to improve the quality of design and not to find a system to transfer the information and knowledge of one designer working at a certain time and in a certain context to another team member. But even if a system is found that thoroughly captures the design rationale, it might not help to build the basis of a design process:

> *While methods of proof might serve their purpose as an aid for laying out or publishing a proof by representing a concise and systematic approach, they cannot serve as the basis of a design process for even a mathematical solution.*
>
> — Pichlmair 2004, p 184

In my experience for developing a good product a designer needs both abilities, the craft of designing and the ability to communicate the rationale in coming up with the design. In doing the craft of design a designer needs to be able to explore, tinker, try and fail, doing it just for the sake of knowing. But once in a while a selection is made and presented to a group of designers or to the client. In those presentations the designers describe their thoughts and their process in finding the selected designs in a rational way. Is the process within the team and with the client then

an interplay of *tinkering and selling*? Is the activity of designing this very interplay? In my opinion it is an important part of it. But I would not dare to say that this is design, too many have tried to define the activity of design.

Tinkering and selling
I use those two terms, tinkering and selling, to describe the two approaches: Tinkering for analysis and synthesis in concurrent interchange. Selling for the rational description of the tinkering phase, this is then in a chronological order.

Before I bring our experience and usage of design rationale in the project fuse, allow me to make a short digression on *bullshitting*.

Digression: Communicating the design rationale = bullshitting?
Bierut (2005) asks the provocative question: What is the relationship of bullshit and design?.

Frankfurt (2005) describes bullshit as:

> *[Bullshit is] not designed primarily to give its audience a false belief about whatever state of affairs may be the topic, but that its primary intention is rather to give its audience a false impression concerning what is going on in the mind of the speaker.* — Frankfurt 2005

He states that when following Frankfurt's definition of bullshit, every design presentation is in part bullshitting. Designers are straight forward in pointing out the functional parts of their concepts, but are, and probably have to be, very unclear about the intuitive parts. Bierut argues that telling the truth, such as *I don't know, I just like it that way*, would not be accepted (Bierut 2005). To be honest, in reality it is often like this.

> *Before I could commit to a design decision, I needed to have an intellectual rationale worked out in my mind. I discovered in short order that most clients seemed grateful for the rationale as well. It put aside arguments about taste; it helped them make the leap of faith that any design decision requires; it made the design understandable to wider audiences. If pressed, however, I'd still have to admit that even my most beautifully wrought, bulletproof rationales still fit Harry Frankfurt's definition of bullshit.* — Bierut 2005

Of course, we do our research, we explore and we try out many approaches, but what triggers a certain solution or why I know I have to follow a certain path and not the other one, I cannot say. In my opinion, in interaction design it is not so much about taste but a bit more about function, and many decisions can be rationally argued. But sometimes a client brings suggestions, which might be valid. Depending on the suggestion, but naturally I try to convince the client with my recommendation

and to bring up some rationale to back it. But when thinking it through after the meeting, I sometimes come to the conclusion that it probably even would not have mattered which solution we agreed on. In this case my rationale is bullshitting.

The same is true when building up the rationale according to the tasks which are stated in the process description. What really happens in a projects looks something like this:

> *When I do a design project, I begin by listening carefully to you as you talk about your problem and read whatever background material I can find that relates to the issues you face. If you're lucky, I have also accidentally acquired some firsthand experience with your situation. Somewhere along the way an idea for the design pops into my head from out of the blue. I can't really explain that part; it's like magic. Sometimes it even happens before you have a chance to tell me that much about your problem! Now, if it's a good idea, I try to figure out some strategic justification for the solution so I can explain it to you without relying on good taste you may or may not have. Along the way, I may add some other ideas, either because you made me agree to do so at the outset, or because I'm not sure of the first idea. At any rate, in the earlier phases hopefully I will have gained your trust so that by this point you're inclined to take my advice. I don't have any clue how you'd go about proving that my advice is any good except that other people – at least the ones I've told you about – have taken my advice in the past and prospered. In other words, could you just sort of, you know... trust me?* — Bierut 2006

As described in our experience on fuse we built up an argumentation line in a sequential order, which roughly followed the description of our process (p 41). In reality many of these tasks happen in parallel, and often ideas which arose even before user research or stakeholder interviews had happened are the ones which are then implemented. At least this was the case for the component selection on fuse. We had this idea very early on, for us it was the natural way in building the prosthesis. We did not know that this was the idea we would stick with, we had to try out many different ones, talk to users, find out more about the context. We needed the time and activities to gain confidence in our idea. But describing this in a process description would be hard, or even not writing a process description at all and just asking the client to trust us (the designer) seems impossible. Therefore we and many others describe an idealized

process and we believe that even the clients need this process to gain trust in our work.

To conclude, a very small part in design presentations is bullshitting, but in my opinion even the best bullshitting will not sell a bad design.

Our experience with design rationale

Let's walk through the design process of fuse: We (interaction designers) play around with the input we gathered from the client and the user research. We do this by sketching and writing scenarios for ourselves or in group sessions. For the moment lets focus on the solitary part. I try out different approaches by sketching, some of them I annotate, for myself, but also to be able to recall it later when I will present some of them to my team members. The sketches get reworked, old approaches get refined, new approaches are tried out. At a certain point (this might be triggered by time constraints or the feeling that I found a few good approaches for which I want to have feedback) I feel confident to present my approaches to the other designers. In presenting these approaches I describe my line of thought. I do not go into detail of all my thinking, and only choose those which I think are helpful for the others to understand my reasoning. My co-designers are doing the same. We come up with new ideas and refined concepts, some of the ideas get discarded. We repeat this process a few times, again the number of iterations is defined by available time, or by the conclusion we draw from input from other stakeholders of the project, in this case our client. The client presentation is prepared more formal and thorough. We build up an argumentation line, so that the client can follow our approach.

In the case of fuse, the rationale was build through a review of the current application, review of the competitors products, the personas, scenarios, *mood boards* and key drivers. Based on those some of the sketches were presented to communicate the client our way of working. We do this despite the fact that we presented and agreed on our process already in the proposal stage, in which we present our process in the form of comparable projects. This has two reasons: (1) Often additional people are in the audience compared to the proposal stage. Even if this is not the case, (2) we learned that our clients and the engineers only know little about our approach to design and our process of working. Because interaction design, or design in general is such a young discipline within software development, a big part of our job as interaction designers is to explain what we do and how we do it. We then show our concepts with scribbles (p 90) and early prototypes accompanied by explanations to put the recipients into the right context. We explain parts which are not

Mood boards
are a means for the designer to explore the emotional landscape of a product or service. Using images, words, colors, typography, and any other means available, the designer crafts a collage that attempts to convey what the final design will feel like.
(Saffer 2006, p 108)

covered by the artifact, talk about transitions and flows which often are missing in sketches, scribbles and early prototypes.

I would describe those forms of presentations – both within the design team and to the client – critique sessions. Depending on my opposite the presentation or selling of the ideas and concepts will tend to be more persuasive or more about gathering feedback and additional input. Within the design team these critique sessions are more open for input and for change, they often end in a group session of exploration and sketching. This differs from presentations to the client. We usually get some first feedback from the client within these presentations, mostly positive and very impressed. But for a more in-depth feedback the client also needs time to let the concepts sink. Usually we have a follow-up meeting to discuss the concepts, questions and feedback. This is completely understandable, because the client sees new and often radical different ideas, which he did not imagine at the beginning. Regardless of the time shifted feedback I would also call those presentations, or better the combination of the presentation and the follow-up meeting, critique sessions.

Such presentation sessions also need to take place between interaction designers and engineers. To get the design built, designers need to understand that we also have to sell the concepts to the engineers and get them motivated to give their best to deliver it to the user. At Razorfish presentations from the designers to the engineers differed often greatly from presentations to the client. Meetings with the engineers were very focused on the hard facts of how something has to get built, but without describing the design rationale behind it. This had the flavor that the engineers only have to implement what the designers are coming up with. The good thing was that over the course of the Audi-project the design/engineering meetings changed to a more open and respectful culture.

On fuse we tried to at least have the lead engineer in all of our client presentations. This was not always possible. Additionally we sat up meetings with the engineering team to walk them through our presentations and findings. We described our design rationale and drivers for our concepts and decisions. After a few meetings those became very valuable critique sessions. The engineers learned to understand, interpret and reflect on our scribbles and prototypes and we learned to understand and question their drivers for the technical decisions. Acceptance of feedback and critique arose among the team after some meetings.

On fuse the meetings were not on a regular basis and by far not as often as within agile environments. But having regular meetings between designers and engineers does not mean that a culture of giving and receiving critique happens automatically. On TempRanger we missed to communicate our concepts and our drivers to the whole engineering team, only the lead engineer of the back end-system and the front-end

developer was present. Also our regular meetings with engineering were mostly with those two. Despite the lead engineer's understanding of our rationale and concepts, he was not able to communicate it to the rest of the back-end team. He was the only one of the engineering team who knew how the product should function and feel.

Consequently an agile environment with its regular meetings does not per se support the understanding of designers and engineers. Regardless of the process environment, plan-driven or agile, effort is needed to build up a common ground in understanding each others needs to be able to provide valuable feedback and input to the various ideas and concepts.

Intention of an artifact

Getting your point across is not limited to the verbal communication in presentations and meetings. Every artifact created throughout the project also needs to be prepared in such a way that the intention of the artifact is transported.

Early in the project many artifacts need to transport that they are not a definitive solution, but ideas which are built to gather feedback. Sketches are a perfect case for this:

> *Even if the designer labored for hours (or even days) over this rendering, and used all kinds of rulers and other drafting tools, it does not matter. The rendering style is intended to convey the opposite, because the designer made this sketch with the clear intention of inviting suggestions, criticisms, and changes.*
> — Buxton 2007, p 106

The visual style of a sketch communicates that it was hand drawn or quickly and effortlessly build with some kind of drawing tool. It is about the transportation of the ideas and concepts and not of the actual style of the artifact itself.

If the language of the artifact itself does not allow to communicate its intention, such as written concepts or personas, they need to be clearly marked as draft. It is important that the other team members are able to understand (by reviewing) whether the artifact is meant to gather feedback or to communicate decisions.

On the other side of the spectrum it is vital that artifacts transport that they hold decisions and that those are the specifications of how the product will look & feel. Often design processes seem fuzzy, subjective and intuitive and difficult to grasp. They often do not follow an approach developed as methodologically as scientific or engineering approaches. But as Wolf et al. (2006) argue, even with a more unstructured approach, a design process is not *black art* (Wolf et al. 2006). *Design has its own*

*internal structure, procedures, activities, and components that are well recognized by skilled designers (*Stolterman 2008) and *there is a rigor and discipline in design* (ibid.).

Four approaches to interaction design
Saffer distinguishes between four approaches to interaction design: user-centered design, activity-centered design, systems design and genius design (Saffer 2006, p 29–42). The first three approaches follow a certain design process, but it is not possible to say anything about the fourth, genius design. Therefore it is uncertain if rigor and discipline are a prerequisite to design in general.

I would not claim that every design approach which aims to develop a concept for a product with a good user experience needs this rigor and discipline as a prerequisite. But I argue that discipline and rigor is needed from a designer to prepare certain artifacts in such a way that they can be transformed into an actual product. This not only includes the final deliverables in a plan-driven environment, such as the specifications, but also all artifacts which hold facts for the time being, such as personas, once agreement on them is established. In agile environments less documentation is done, and the artifact is usually evolving over time. Nevertheless also here certain artifacts, or parts thereof need to be exact and concrete.

Jeff Patton reports an interesting comparison to architecture and the responsibility of the architect from a workshop with user experience designers and software architects:

> *But ultimately the architect is responsible for the success of the building. […] Because he's responsible for the success of the building, he doesn't throw his designs »over the wall,« but stays with them throughout construction and into operation.*
> — Patton 2006

This is not the case with software development, and I would not argue that this is the right way to go, and that the interaction designer alone is responsible for the success of the product. But I hope that in the future interaction designers take on more responsibility and stay in the project over the whole course of the project. What the interaction designers can do already is taking responsibility for the artifacts, especially the specifications they deliver. Someone has to decide on the details – and if the designers are not doing it, the engineers have to do it, because otherwise they cannot implement it.

In my experience taking this responsibility and delivering specifications in such a way that they are easy to implement by the engineers earned us credibility and we were taken seriously. By easy to implement I am not talking about the actual content of the specifications, it still might be a challenge to find a good technical solutions for it, but about the form of the specifications.

10.5 Experiences from different roles in a team

In this chapter I talk about some of my experiences from different roles within small and large teams in various projects. The descriptions focus on how we dealt with user experience, who was responsible for it, what worked good, what did not?

Project: møbelhus

This project is not included in the chapter *Projects* (p 24) with a short abstract, therefore I briefly describe the project layout here. møbelhus is a client/server based administration software for production planning, controlling, job costing, purchasing and order management for a mid-sized company. It was designed and developed over the course of two years from 1995–97 by myself. Commissioned directly by the owner of the company, he was the only person I needed to convince of the fact that I also like to involve the users of the application. The good thing being a one-man team is that no knowledge gets lost between designing and implementing the software. On the negative side was the missing sparring partner and that I might have constrained myself too much, because I knew I had to implement it. Nevertheless the application was highly appreciated by the client and by the users and is (after some adaptations) still in use.

Project: k2

Although I was not part of the interaction design team on k2, I was part of the implementation team and responsible for the technical architecture and the database model. Hence this experience is from the viewpoint of an engineer, although with a very strong design mind-set.

k2 was appointed in 1999 to the uid-lab at the Vienna University of Technology. k2 is a database client for the biggest staffing service in Austria. Usually database clients are driven by the structure of the underlying database design. Here the focus of the design was on the users needs in the special situation of handling many applicants with varied backgrounds and on the task of how to find and choose the right one efficiently in a multi user environment across different offices.

I came onto the project after the design phase was finished. During the implementation we ran into technical problems. This had two reasons: (1) At the beginning the client wanted us to use their database model and build the application around it. The problem was that the model was neither very flexible nor performant. Because their old database was never really used, the problem of data migration was not an issue and we were able to convince the client to let us build the database model from scratch. (2) The design team had only little knowledge about the limitations of the technical environment (the environment was preset by the

client) and therefore some of the concepts could not be implemented the way they have been designed. Additionally, because of a limited budget for the design phase, some of the concepts were not thought through in detail.

For all those situations we – the designers and engineers – had to come up with ad-hoc solutions to be able to deliver the project to the client. From the point of view of a software engineer, not knowing how some features were supposed to work, was sometimes tiresome. On the other hand, the collaboration with the designers worked excellent. Over time we figured out for which situations we engineers could decide on design decisions and for which we needed to consult with the designers. In my opinion this was only possible because of the general technical understanding and acceptance of constraints of the designers, and the design thinking of the engineers. Mutual understanding was key to adapt the innovative concept in such a way that it could be implemented and still was an innovative application with the user in mind.

A detailed description on the design of k2 can be found in Purgathofer's habilitation treatise (2003, pp 189–231).

Project: Audi website

Before describing my experience on the Audi-project I need to give some background information about the organization of Razorfish for a better understanding.

Razorfish was organized within four networks:

- **Value:** Responsible for creating lasting relationships with the client and delivering projects on time and within budget. Exemplary roles of this network are project manager and client partner
- **Strategy:** Responsible for developing strategies that stay valuable over time. Exemplary roles of this network are strategist and brand strategist. The user research was also included in this network.
- **Experience:** Responsible for designing an innovative, high-quality user experience. Exemplary roles of this network are information architect, interaction designer, visual designer, usability analyst and writer.
- **Technology:** Responsible for the quality and delivery of all technology artifacts. Exemplary roles of this network are technical business analyst, technical architect, data modeler and programmer.

Projects at Razorfish were staffed with people out of the four networks, as was the Audi-project. The core team consisted of one person of each network.

Within the experience network we had people with different backgrounds, such as (graphic) design, library science, writing and computer science. The level of the technical and business understanding was very diverse. Some of the members of the experience network had no technical understanding at all.

We had learned from previous projects that it was important that the experience lead has to have some technical understanding, to be able to communicate with the engineers. Business understanding was good to have too, but in our experience not as critical as the technical understanding. On one of those projects the experience lead was staffed with a visual designer, who did a fabulous job in dealing with the client and leading the team to come up with innovative concepts, but who had little technical understanding. This peaked in presenting concepts to the client, without involving the engineers beforehand and checking if we as a team were able to deliver those concepts. Conflicts between the experience and technology team were at that time common practice.

On the Audi-project the experience lead in the beginning was a visual designer, but with a strong understanding of technology. My role in the project was the one of an information architect. Later on I took over the role of the experience lead.

In the role of the information architect my task was to get the different interests of all different departments within Audi onto one website. The previous website pretty much used the internal structure as the navigation concept. This was not very useful for the users, therefore we came up with a navigation concept which caters to the needs of the users. This was also in the interest of Audi. But it also meant for Audi that departments which never had worked together before, had to share responsibility for certain sections now. We had to deal with the internal conflicts and power struggles of the different departments to get to a common understanding on what could be placed where within the website without neglecting the interests of the user.

Already in the role as an information architect I often acted as liaison to the technology network, because of my background in computer science. By taking over the role of the experience lead I got assigned the role of a communicator within the experience team, between the different networks and the client.

Were we, the experience network, the owner of user experience? The description of the network stated it. Also the general understanding in the company was that we own user experience. But in my opinion we did not own it. Of course we were responsible for the concept and all the visible elements of the product. But coming up with a concept would not have been possible without the input of the other team members or without

the client. Even more important, bringing the product – and with it the user experience – to the user needs everyone of the team.

Certainly we had a big stake in influencing the team and in guiding it to a more user-centered and holistic approach. Being able to communicate in the languages of both worlds, design and engineering, established an environment of real acceptance and not just formal *we have to ask the [replace by other party]*. In my opinion, only because of this environment we were able to deal with technical issues under tight time constraints at the end of the project – without neglecting the user experience. Members of the experience network were involved in the decision process on how to change features or even which features to postpone until the end of the project.

Project: fuse

On fuse we were in a different situation. Compared to the Audi-project on which everything was developed in-house at Razorfish, here we were an external company hired for doing only the interaction design. Being an external company had both advantages and disadvantages.

Let's start with the disadvantages. We were hired by product management and thus the engineers were not convinced that we would contribute much to the project. At best they saw us to beautify their current application and to combine them nicely with some sort of a launcher. They had already worked a few years on the applications and did not believe that we were able to pick up the specifics of the medical context in such a short time. Furthermore they questioned our methods of user research. We worked hard to overcome this barrier by gaining as much knowledge about the medical context as possible. We showed respect to their existing knowledge, and asked them for meetings to share it with us. In addition we explained our approach and gave reason why we came up with certain concepts. Being able to understand and speak their (technical) language helped tremendously. It took some time to overcome this barrier, but the relationship improved from meeting to meeting. A few weeks in the project the engineers started trusting our holistic approach and the way we involved them in the creation of the overall concept.

There are also advantages when acting as an external resource: namely being independent of any stakeholder involved. Of course, product management was our contractee and we were accountable to them. The engineering team was scattered over two locations, both developing their own applications. Naturally they had different opinions on which basis the new software suite should be build on. The branding department did not like the look and feel of the old application, but did not have a say in software development until we came onto the project and asked them for brand specifics. Of course there were the typical power struggles and very

diverging viewpoints among the different product managers and sales people. Because of our independence we were able to act as a filter – or lets call it as a *catalyst* amidst the different parties.

Unfortunately, we were not able to stay in the project during the implementation phase to deal with upcoming issues. Therefore it is even more important to transfer the way of design thinking to the team, Vanka (2007) calls this *raising the design IQ within in the team*, and to also prepare the delivered artifacts in such a way that they make it possible to deliver the developed concept to the users.

Again, as on the Audi-project, we were not the owner of the user experience, but we were the animating spirit on the team, first by convincing the client to trust us on the user-centered holistic approach and then by transparently applying it.

Needless to say that in all of the projects outlined above the capabilities as stated in the previous chapter were important, but in my experience being able to collaborate with and integrate all team members into the process is equally important.

10.6 The catalyst role

For Anderson et al. (2005) *Who owns user experience?* is the wrong question, the question should rather be *How do you manage the kind of collaboration that leads to real change?*

Based on my experience one of those tools for helping to manage the collaboration better, is the role of a collaborator, or I would prefer calling it the *catalyst role*.

McMullin (in an interview with Merholz 2007) calls this role *facilitator* and argues that the designer would be a good fit for it:

> *The reason that a designer ends up being the facilitator is because all those same skills that we've cultivated – in empathy, in listening, in observation, in synthesis, in actually creating tangible artifacts that people can reference and discuss – all of those same skills that we would use in a user-centered perspective, if we pivot 180 degrees and then look at the business and look at the team, we can use that same skill set and many of the same methods to facilitate a consensus and get people talking from their different frames of reference so that they can actually articulate what's important to them.* — Merholz 2007

In my opinion everyone on the team can take over the role of the catalyst. Michael Lopp talks in a conversation with Jeffery Zeldman about his job

as a manager at Apple and describes his role as being the synthesizer on the team, who translates between designers and engineers by knowing and speaking both languages (Zeldman & Lopp 2008). My own experience as well as that of Ashley & Desmond (2005) showed that this role is often taken by an (interaction) designer with a broad understanding of other disciplines. On larger teams the catalyst role could even be a role dedicated to one person. On smaller teams this would mean that the relevant team members – usually they who collaborate frequently – should have the catalyst capabilities.

IDEO
is one of the best known design consultancies for product, service, and digital experience design. They were among the first *to integrate the design of software and hardware into the practice of industrial design* (Moggridge 2006).
http://www.ideo.com/

T-shaped people, as Tim Brown (2005) of IDEO calls them, would perfectly fulfill the role of a catalyst:

> *[T-shaped people] have a principal skill that describes the vertical leg of the T – they're mechanical engineers or industrial designers. But they are so empathetic that they can branch out into other skills, such as anthropology, and do them as well. They are able to explore insights from many different perspectives and recognize patterns of behavior […].* — Brown 2005

Spool suggests to interchange team members between different disciplines, as guest members, over a short time frame, to exchange knowledge and learn each others methods and tools (Spool 2007b). This is an excellent method to gain the capabilities needed for the catalyst role.

10.7 Balanced teams

As stated above the widely held opinion is that the whole team owns the user experience. How should such a team be laid out?

Berkun suggests to bring in an interdisciplinary view to the project team in order to be able to interact across boundaries. A project should be balanced between the three perspectives by decisive power (Berkun 2005, pp 44–52):

- **Business perspective:** This view is concerned with the profit and loss of a project, and it covers accounting, sales, competition, marketing, management.
- **Technology perspective:** This perspective is concerned with the construction of the software and its maintainability.
- **User perspective:** This perspective is concerned with who the users are, how they are currently doing the tasks, and what they definitely and maybe need from the software. This is the most important of all three perspectives, but usually the weakest in many organizations.

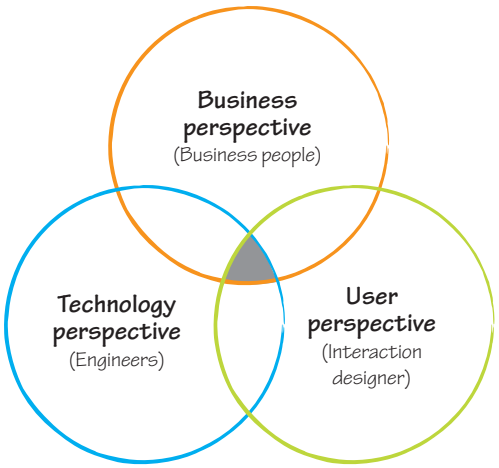


Figure 13: The three perspectives of a project (Berkun 2005, p 50).

This view of projects also follows Boehm's definition of software engineering: *the application of science and mathematics by which the properties of software are made useful to people.* (Boehm 2006, p 12). With the phrase *useful to people* he includes the relevant sciences (labeled perspectives in figure 13): behavioral sciences for the user perspective, management sciences and economics for the business perspective and computer science for the technology perspective.

Those three perspectives are very common in many projects, the problem is that they are unbalanced in terms of decisive power. Vanka reports that in typical projects engineers outnumber designers by 100:1. Even at Microsoft (where the ratio is far better towards the designers) it is still 42 engineers to 1 designer (Vanka 2007). This imbalance is often the reason why the user perspective is so weak.

For a balanced perspective the power ratio should be 1:1:1 for each of the perspectives: business to technology to user (Berkun 2005, p 52). On a team this balance with a raw count of team members per perspective will hardly ever be possible. I have never been in a project in which the designers outnumbered the engineers or even came close to be on par. But as Berkun argues, *the raw numbers of people don't define how much power they have* (Berkun 2005, p 52). Important is that all three parties have equal power over the project, even if there are only 2 business people, 3 designers and 10 engineers. Thus the power is democratic within the three perspectives and not the whole team. Balancing the three perspectives is one, if not the most significant, task of the project manager:

> *[…] a manager can compensate for any natural ratio by granting power to those who should have more influence on the project. And because the nature of a project changes over time, different perspectives should have more power at different times.*
> — Berkun 2005, p 52

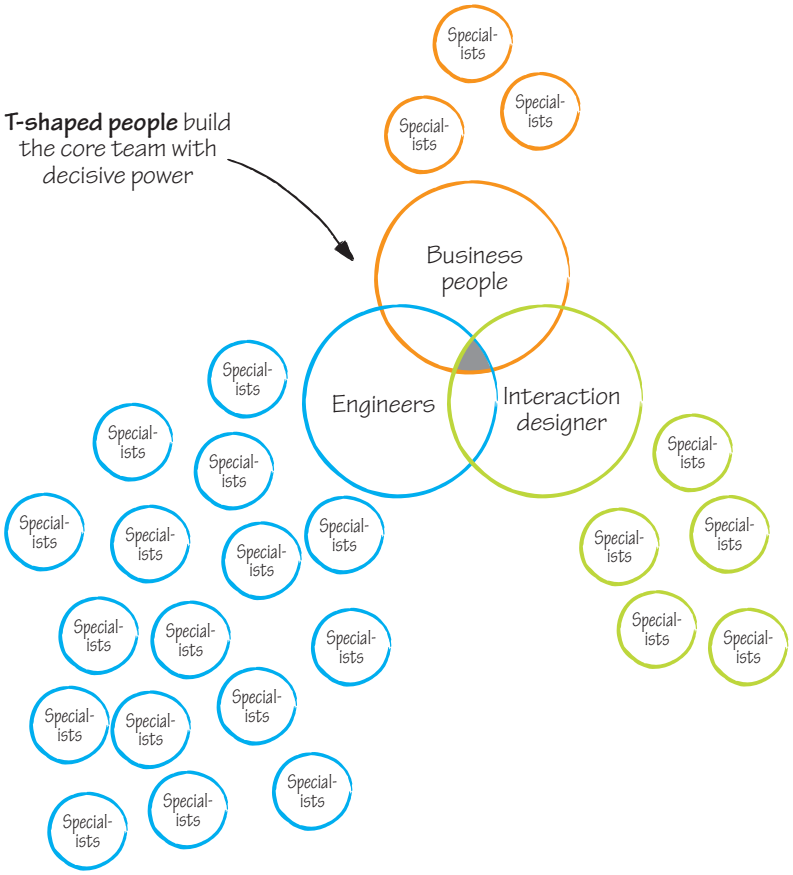


Figure 14: Balanced core team of T-shaped people. Everyone on the core team should fulfill the characteristics of the catalyst role.

In a plan-driven environment this would mean that at the beginning of the project the power of the user perspective is (or should be) higher. During implementation this may change towards technology, but still the designer is involved in the decision.

Often this balance is established through a core team, with a member of each perspective and equal power. On the Audi-project we worked in such a setting with a positive result. We worked with a core team of four parties, but those could easily be mapped onto the perspectives: the value and strategy network onto the business perspective, the experience network onto the user perspective and obviously the technology network onto the technology perspective. Over the course of the project the power shifted slightly from strategy and experience (at the very beginning) to experience, to experience and technology, lastly to technology (at the very end). Except for strategy, which handed over its stake to the value network, all networks stayed on the project over the whole duration and formed the core team with equal power. With this layout we were able do deal with critical situations, which we had plenty of, and make tough decisions without loosing sight of any other perspective. Miller (2005) describes a similar team layout on her experience with the development

of SketchBook at Alias in an agile environment. The core team was formed of the product manager, the development manager and the design manager – herself.

Ideally the core team would be staffed with T-shaped people, acting and thinking in a designerly way, as catalyst between the different perspectives (see figure 14).

11 Artifacts

In this chapter I cover artifacts, which are used in projects and can act as a boundary object (p 19) and as inquiring material (p 17) between designers and engineers. They are not the final output of design (which is the application itself) but function as *means for inquiry* (Gedenryd 1998, p 149) for the whole team. The driver for the selection is based on our experience in projects and the impact they had to bring a more holistic view of product development to the team.

I will only give a brief description of the artifacts, which is needed to gain an understanding for all participating disciplines. For a more elaborated description and especially for methods and best practices on how to develop those artifacts the relevant literature should be consulted.

The focus in this chapter is on the usage, sometime possible usage, of the presented artifacts of the different team members in a project.

11.1 Personas

Personas (artificial users profiles) are a commonly used method within the design community. The concept of personas has been around for a long time, although without using the term *persona* (Norman 2004a, Pruitt & Adlin 2006). They were popularized within the field of software development by Cooper (1999) in his book *The inmates are running the asylum* in 1999.

Why personas?

The *elastic user* (Cooper 2004, p 127) – everyone thinks he knows the user, and everyone tries to design for him. But more often than not the user, as an anonymous person, is a cover for the ideas and wishes of oneself. How often have I heard: *But the users think that …*, and then the user gets pulled in the direction of the team member's ideas.

It is valid to design after one's own ideas as long as it is not covered as representing it as the users' wishes. Personas are a tool to give these anonymous users a face and to encourage team members to distinguish and also articulate their own ideas apart from the users' view.

What are personas?

Cooper describes personas as:

> *[…] a precise description of our user and what he wishes to accomplish.* — Cooper 2004, p 124

He argues that if one designs for a large audience it is better to design for a single user than to accommodate everyone (see also Miller 2005). The

hans-dieter strunke, 46
primary user

the meister

meister and head of the department »prosthetics lower extremities« at sanitätshaus monke in stuttgart, germany

»da paßt noch nicht alles«

hans-dieter is one of four meister in his branch. his day is split between consulting patients and doing the paper work — which is ever increasing by the day. in the patient treatment he does the consulting, makes sure that the socket fits and selects the right components for the prosthesis. usually he orders the test-socket at otto bock, building the definite socket and the prosthesis is done by his coworkers. for the administrative work he is using the computer more and more, especially for patient management.

hans-dieter's goals

- i want to make sure that everything is right, but i don't want to spend too much time in front of the computer
- my workflow works best for me, i don't want to change it just to use the computer
- once i hand over the job to the assistant i want to be sure he gets everything

company information
- provides services in prosthetics, rehab & ortho shoes
- 60 employees, 3 branches
- he works in the head office with about 40 employees
- the 2 branches are smaller, each with 8 and 12 employees respectively

psychographics
- late mainstream — skeptical, adopting only after a majority have done so
- he only jumps onto new things if he really sees that it's worth it

computer proficiency and usage
- beginner
- daily uses sanivision for patient management
- infrequently excel for calculations
- daily email but only gets about 3–4 emails/week
- digital camera for patient documentation
- at home he surfs the internet infrequently on his son's pc, mainly for information about traveling, he also got an internet account from his bank but does not use it

computer equipment used by him
- pc for himself, about 1 1/2 years old, with 17-inch monitor, resolution 1024x768, windows xp
- tt-design, tf-design, sanivision
- ms office, ms outlook, ms internet explorer, digital camera software
- laser printer, digital camera, l.a.s.a.r.
- shared laptops, 2 years old, windows 2000, 1024x768, tt/tf-design, c-soft, myosoft; those laptops are shared amongst the employees if they need it directly with the client
- at home his son has a pretty new pc

product relationship
- occasional user of tt-design
- ~1.300 patients/month treated in all branches
- ~50 patients/month treated by him altogether
- 1 patient/month treated by him with tt-design
- 8-10 tf-sockets/month ordered by him via fax — he does not use tf-design
- 70% of all the tf-sockets are ordered by fax, only 30% by email

attitude toward product
- software is too complex, he thinks he is faster with plaster cast

Figure 15: An example primary persona from the project fuse.

idea of personas sticks to this concept: to design an interface for each primary persona (see below for different persona types).

Personas

- are a portrait of a typical user ideally based on user research data and input from the client. If user research is not possible so called ad-hoc personas might suffice. In this case the team together with the stakeholders construct the personas.
- are hypothetical. They are not actual people out of the user research, but based on user research.
- are archetypical and not stereotypical.
- represent important demographic data.
- live in a social context.
- have characteristics and goals.
- are *alive*.

It is nearly impossible to cover the target group with one single persona. We usually create between four and eight personas, depending on the complexity of the project and diversity of the target audience. But for which of those personas should the product then be built? Cooper & Reimann (2003) suggests to use different types of personas: primary,

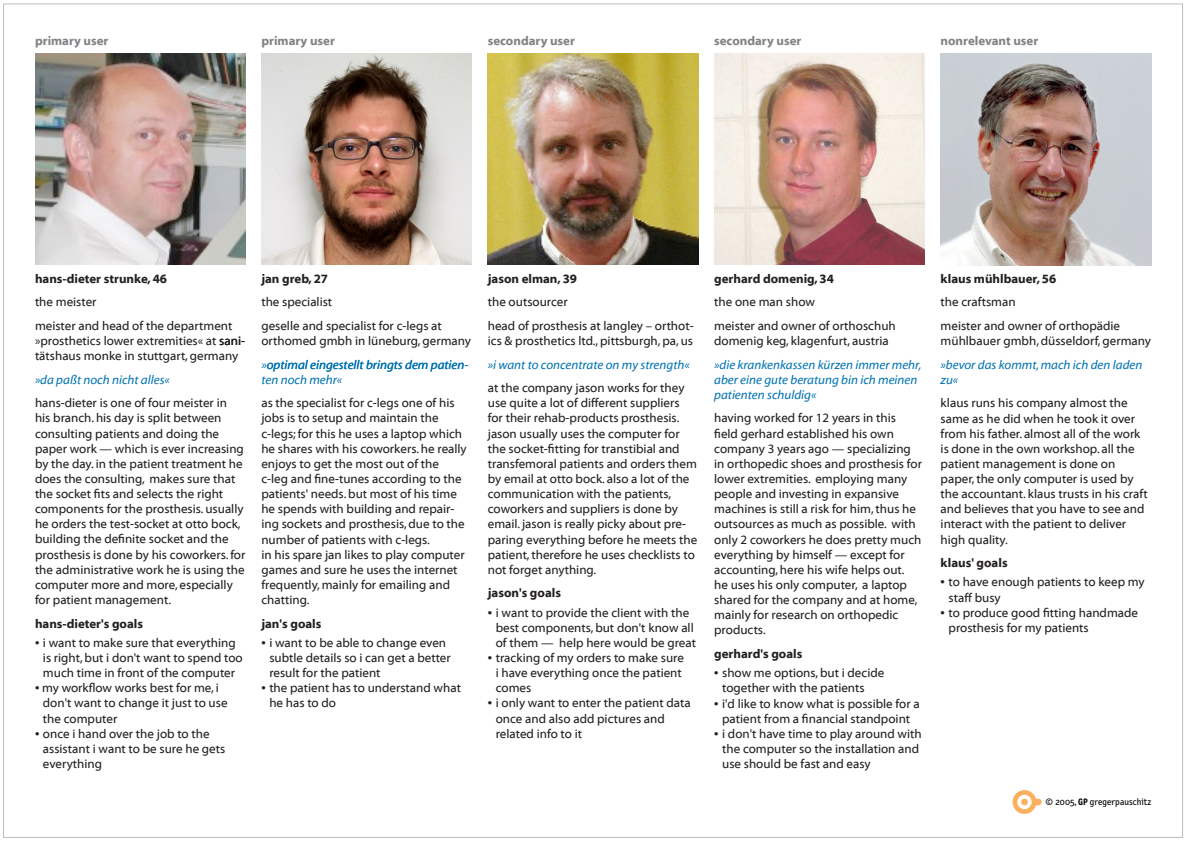


Figure 16: Overview of the different types of personas for fuse.

secondary and negative personas (he even describes several other types, but these are the ones we usually use). The primary persona (usually one, but in larger projects sometimes several are needed) is our most important persona. Each of the primary personas gets his own interface. This means that, if there are more primary personas, also more interfaces are needed – or at least dedicated parts of it. Secondary personas should be generally happy with the interface for the primary persona, but they have additional needs which should be integrated without getting in the way of the interface for the primary persona. Sometimes also negative personas are helpful, to clearly state the borders of the project. Those are personas we are explicitly not designing for.

Critique on personas

Personas are not only seen positive. Recently critique was issued. Portigal (2008) states that personas are *misused to maintain a safe distance from the people we design for*. Personas were intended to achieve quite the opposite. But it seems that in reality most practitioners use ad-hoc personas. Spool (2007a) conducted a field study of companies who are using personas and discovered that only 5% of them are basing their personas on user research in the field, 95% are basing their personas on

internal knowledge. The ratio of those two should of course be the other way around.

Using ad-hoc personas should be the exception, but might still sometimes be useful. It often is not possible to interact with actual users. This might be because of confidentiality reasons, budget constraints, or even internal team hierarchy. Sometimes only the sales people are allowed to talk to the users – or better call them customers, because they usually very seldom talk to actual users. But even in this case it is better to come up with ad-hoc personas and use them throughout the project, than not having personas at all. Norman goes even as far as postulating that personas do not always need to be accurate in every detail, they only need to reflect the target group (Norman 2004a). They help the team to question whether their decisions are still meeting the audience it was intended for. This is especially helpful when new team members are brought on to a project and usually bring in new ideas and viewpoints, which is good, but sometimes also dangerous.

Personas are more a communication tool (Cooper & Reimann 2003, Cooper 2004, Norman 2004a, Pruitt & Adlin 2006) than a tool for designers. A good designer does not necessarily need personas to be able to set himself into the shoes of the user. But they are helpful more to junior staff and to other team members, whose main responsibility is not to act constantly out of the users perspective. Even if personas do not represent the users completely and *don't talk back* (Fried 2007) as real people do, they are a tool to better communicate real users' needs and what they would answer – personas start a discussion process. This would not happen if personas do not exist.

But why not using the real users instead, as in agile environments? This is often not possible or not allowed by the management. Even in agile environments one can only talk to a handful of users and then this feedback comes uninterpreted and unreflected into the process. Personas are a tool to present the whole team the knowledge gathered in a condensed form.

Our experience with personas

Depending on the project we try to convince our client to use personas. Personas would probably be a good method in every project, but project management constraints (such as time and budget) often do not allow it. Furthermore, in many projects we are not allowed to talk directly to the users – be it for confidentiality reasons or to ensure that we do not raise hope in the customers of a soon to be delivered new version of the product too early. In this case ad-hoc personas could be used.

As an example of our experience with personas I use the project fuse and also some insights we gained from the utilization of personas in teaching projects with students.

As on many of our projects, also on fuse, the method of personas, or even of user research was new to our customer. We pointed out the possible advantages and explained our approach in detail. They were willing to give it a shot, although still skeptical. The user research, interviews and workshops with all the project stakeholder proved to be important for us in order to gather an understanding of the different aspects of the project. We were able to directly talk to users and observe them in four medical workshops in Germany and Austria, and to conduct phone interviews with a medical technician in the USA. The research brought up important new facts for us. I just bring one as an example: In contrast to the intended usage of the new software, the medical technicians do not use a computer in the presence of the patient. Rather the information is still gathered on their custom made paper forms, which more or less all follow the same layout. The data of all patients is collected throughout the day and then transferred to the computer either by the medical technician himself or by an administrative assistant in the evening. This was completely new to the whole project team. The old version was designed to be directly used in the presence of the patient.

We summarized all this information in the form of many scenarios and personas and presented a first version to the client. An important part of the presentation was the description and explanation of what personas are and how we will further use them in the project. In our experience it was new and confusing to the client that we narrow down the target group to a few archetypical personas. The reluctance to not design for all people was very high. This especially emerged when we presented a negative persona. In this case Klaus, an older medical technician, who to retire soon does not have any computer literacy at all. The interesting thing was that this requirement was spoken out clearly by the client from the very beginning on. But expressed as a persona it made clear that we exclude certain users. For the client the requirement was still valid, but instead of using a negative persona, we used the term non-relevant persona, to soften the meaning. But it was important for the project to clearly speak out this fact and to get all on the same track.

Another surprise for many team members on the client side was that none of the personas matched any actual people we had interviewed. On fuse this is especially noticeable, because many of the product managers and engineers knew the users of the visited medical workshops from their collaboration with them in previous projects. But the discomfort at the beginning of not using real people quickly faded away after a few meetings in which we discussed and finalized the personas.

Once we gain agreement on the personas and their characteristics, the detailed personas (figure 15) are printed on A5-format sheets to give it a card-like feeling. The overview sheet with a summary of all the personas

(figure 16) is printed in A4-format. All of the print outs get laminated, so that they are not easily tucked away in a project folder and thus never looked at again. It also makes them more durable, consequently averting accidents (such as coffee spills) which would prevent people from bringing them to meetings. At least each department gets its own deck of personas. On some projects we even hand out decks to every engineer on the project. From that point on the personas are brought to every meeting, internal and external, and the overview sheet is put in the center of the table. This encourages us all to use the names or nicknames – all of our personas have meaningful nicknames, such as *the meister* or *the one man show* – of the personas and to integrate them into our meetings. So, when someone argues about a feature he would clearly have to distinct between his own thoughts and what one of the personas might think. The knowledge gained in the user research was in this way easily shared among the team members throughout the project.

Some time into the project even the engineers started to call the personas by their names (instead of their roles) in their conversations. A few months into the project I got a phone call, and a developer described some new possibilities he had come up with and how this would help Jan, *the specialist*, to better adjust the fitting of the prosthesis. We did not expect this to happen, because the engineers were unsupportive at the beginning and did see personas as an unnecessary part of the project. Their understanding of us was that we were only allowed to make their product prettier, but not to change much of the functionality – as this would most likely also have an impact on the code base.

The experience I gathered while teaching personas to students goes in a similar direction, but lies more on personas as a design tool in the classic way. Here personas are more a tool for the designers, to help them become the user for whom they build the product (Pavese 2007). Personas support them in role playing and also in communicating about the target group. But it also takes a while for the students to understand the full qualities of personas. Over the course of three semesters, they accepted personas from a mere deliverable to an effective tool in projects. After an introductory course in interaction design in the first semester, their task in the following two semesters is to build a product (most often a website) from the very beginning to delivery in a team of five students. They learn to integrate the personas through the course of the whole project.

Personas as a boundary object

When introducing personas to the engineering team they often get mixed up with actors used in conjunction with use cases within the unified process. The formal definition of an actor is: *someone or something, outside*

the system, that interacts with the system (Leffingwell & Widrig 1999, p 42). Thus an actor is someone who uses the system, someone who provides to or gets information from the system, someone who maintains this system or another system who uses and interacts with this system (Jacobson et al. 1999). The important part about actors is that they are a role and not an actual person. A user or (with the method of personas) a persona can act in many different roles.

The mixing up of personas and actors is both good and bad. The good part is that personas together with actors perform as a boundary object. The method of personas is mostly unknown in the engineering community. Use cases are only rarely done by designers, this is usually the job of the business analyst or technical analyst. By letting personas perform in different roles, as an actor, they become more vivid, instead of the cold and impersonal drawing of the actor in the form of a stick man. If personas are based on user research, the link to use cases is established. With this connection of personas to actors and its transformation into the more analytical world of software design, everyone on the team can better visualize for whom they design the system. A basis for a more holistic project view is set. Engineers integrate the actual users, based on user research in the form of personas, into their use case model. Designers, on the other hand, get an insight into the process of how engineers are modeling the implementation of the system.

Are personas also a useful method within agile environments? This depends on how often one can reach the actual user rather than only the customer (which happens too often in agile projects). It also depends on how diverse the user base is and if this diversity can be accommodated with the users available. As described in the chapter *Process* (p 32) often access to users is limited and is not possible in every iteration. Personas are a useful method to summarize the findings, and they act as stand-ins for the user in the user stories.

Indeed, we should not forget about the bad thing, the misuse of personas (Portigal 2008). That personas are only seen as a more elaborated explanation of an actor, and maybe even driven by a system centric view and not being based on actual user research. Even more importantly, that personas, and with them also the users, are not the driver of the system. In this case personas would only act as a cover for user-centered design. But this is the risk with every tool or method: used the wrong way, it produces poor results. I am willing to take this risk, because in our experience personas have a positive impact on projects, and help focus the team on a more holistic view of the whole situation.

11.2 Scenarios

Scenarios and stories
I will use scenarios and stories interchangeably. Some argue that scenarios are more abstract than stories. In my experience they are often used interchangeably. By looking for artifacts which are used by different communities it will always come to different naming conventions. One of the goals of the catalyst kit is to encourage team members to talk about such issues and agree on a certain naming for the current setting.

Scenarios are a widely used tool throughout the project life cycle. They are applied by many communities, such as strategic planning, human-computer interaction, requirements engineering, and object oriented analysis and design (Go & Carroll 2004). Those four communities comply with the three perspectives used in this thesis (cf. chapter 10.7, p 68) as follows:

- Strategic planing – business perspective.
- Human computer interaction/partly requirements engineering – user perspective.
- Partly requirements engineering/object oriented analysis/design – technology perspective.

Scenarios were first introduced in 1962 in strategic planing, by the mid-1980s they were used in other communities as well (Go & Carroll 2004).

What are scenarios

Scenarios are descriptions of people and their tasks (Carroll 2000) and are widely used within different forms throughout the team in a project (Go & Carroll 2004).

> *Scenarios are paradoxically concrete but rough, tangible but flexible […] they implicitly encourage »what if?« thinking among all parties. They permit the articulation of design possibilities without undermining innovation. […] Scenarios compel attention to the use that will be made of the design product. They can describe situations at many levels of detail, for many different purposes, helping to coordinate various aspects of the design project.* — Carroll 2000 cited in Cooper & Reimann 2003

They might be presented in many different ways, such as narratives told, textual stories, storyboards, video prototypes or low-fi prototypes (Go & Carroll 2004).

Designers use scenarios in the form of stories told and discussions to discover new grounds (Buxton 2007, p 262). Often also in the form of role playing by taking the part of a persona. In written form to transfer the knowledge gained during user research to the rest of the team (Cooper & Reimann 2003). Here they are a mixture of the currently applied procedures and the expressed wishes of the users of how a future system should or could work. Scenarios typically focus on happy day scenarios and do not deal with all exceptions.

Scenarios versus use cases

The most popular usage of scenarios by engineers in plan-driven environments is in the form of use cases. A use case is defined as:

> *A use case describes a sequence of actions a system performs that yields a result of value to a particular actor.*
> — Leffingwell & Widrig 1999

A use case describes all possible paths – in contrast to a scenario, which only describes a part of the possible paths of events. Additionally, use cases are described in a formal way, scenarios are written in an informal way (Jacobson et al. 1999). Scenarios describe not only the behavior and functions of a product, but also the priority of the functions and how they are presented to the user (persona). Use cases on the other hand do not describe the precise behavior of a system. Use cases do not state how the system should interact with the user. Use case models give an excellent overview of the whole system, but do not transport the priorities of the different cases (Cooper & Reimann 2003).

Scenarios versus user stories

In agile environments user stories are used to gather the requirements, and are defined as:

> *One thing the customer wants the system to do. Stories should be estimable at between one to five ideal programming weeks. Stories should be testable.* — Beck & Andres 2004

> *Stories need to be of a size that you can build a few of them in each iteration.* — Beck & Fowler 2000

A user story is a customer requirement worked out within one or two sentences from a user perspective. It describes how a user might use the product, and is written in natural language. They are limited in length and depth by the estimated development time. They are created together with the customer by the development team, but are owned by the customer.

Compared to scenarios, user stories are usually much shorter and focus on exactly one detailed function. Scenarios do not have the limitation of being implementable in a certain time frame. They are described in depth and length appropriate for their intended use.

Our experience with scenarios

I cannot remember one single project on which we did not use scenarios. We usually use them in many different ways and in different phases of the project. At very early stages of the project – often during, and sometimes even prior to user research – we discuss different scenarios in the form of stories to gain a better understanding of the users needs and their contexts. This is very much in the sense to which Buxton (2007) understands stories:

> *The key to stories […] is to help discovery. […] it is the discursive element – the back and forth playing with the story among the design team or the audience – wherein the insights are found and where the value lies.* — Buxton 2007, p 262

In my experience this kind of scenario development is very valuable within the design team, but it might be both too artificial and too far fetched for the engineers. Therefore we typically do not integrate them in this process. Storytelling is a form of reflective practice, similar to sketching, but in a group setting. Mostly we also heavily sketch during those sessions.

Depending on the project we summarize the scenarios in a more formal style and get input from the client and engineers. Those scenarios are usually already accompanied by sketches and first low-fi prototypes. On smaller projects we generally do not write down the scenarios in a formal way. We just use them to come up with sketches (storyboards) and low-fi prototypes. Then we present them to the client and engineers. In this chapter I describe the formal usage of scenarios. The informal usage in the form of presentations and low-fi prototypes are covered in the chapter *Getting your point across* (p 55).

Scenarios within the project fuse

On fuse we combined the background information we gathered during the user research with persona-based scenarios (Cooper & Reimann 2003) of the imagined application (see figure 17). This approach helped us to clarify our understanding of the current situation and thus to present our concepts and ideas. We learned that especially the provided background information (in contrast to the scenarios) allowed the product managers of the different products to gain insight into the needs of users of the other products. Hence all stakeholders and team members got a better overview of the situation and the challenges of bringing all different applications to one single platform.

With this approach we were able to show the engineers that we understood the setting and as such we gained some credit. They were still skep-

tf-design: background info

measuring the patient's residual limb
the patient and the ortho-technician discuss the situation and possibilities. if the patient got amputated recently or her situation is changing rapidly for other reasons then she will probably get 2–3 sockets in the first year to accommodate for changes of the limb or the weight etc. otherwise the patient will get a new socket/prosthesis about every 2-5 years. sometimes a few relevant pictures of the residual limb are taken. with some special tools the ortho-technician now takes all necessary measurements — eg the length of the residual limb and its circumferences at several intervals together with those intervals as well as the length of the socket — and notes them down on a printed form. in addition the patient's weight and her mobility grade are assessed.

importing measurements, rectifying and tweaking resulting 3d-model
[either with or without the patient, either done by the ortho-technician or her assistant]
the ortho-technician enters the measured data into the software with the patient either on the spot, in his head or on a picture. from his experiences he will apply a certain reduction to the circumferences. he then takes a look at the computed 3d-model of the resulting socket to check if everything is alright, which might involve taking some measurements in the 3d-model to compare them with the measurements he took on the patient. for best results he will tweak the 3d-model a little bit, eg by changing the angle of flexion and adduction or by adjusting the distal shape. rarely he compares this socket with previous sockets of the patient.

assembling of the complete interim prosthesis and ordering of parts
the ortho-technician virtually assembles the complete interim prosthesis according to the patient's weight, mobility grade and personal preferences and orders the test-socket along with those parts that are needed but not in stock.

delivery of the test-socket
after 2-5 days all the ordered parts are delivered, including the test-socket.

trying the test-socket
the test-socket is transparent to allow for visual examination of the fit. the patient puts on the test-socket, which the ortho-technician modifies, mostly with a heatgun, until it fits properly. then the prosthesis is assembled — which takes about one hour — and finally connected to the socket. the patient puts on the complete prosthesis and walks around to test it. inevitable changes are incorporated.

testing the interim prosthesis with the test-socket at home
sometimes the patient takes the interim prosthesis with her to test it under real condition over a longer period.

modifying the prosthesis and decision on cosmetic details *
after about a week the patient returns. she can now give detailed feedback about the fit of the socket and the other components of the prosthesis. the ortho-technician incorporates the feedback and modifies the prosthesis. they decide about the cosmetic details and the ortho-technician orders all needed parts which are not in stock.

definitive socket including cosmetics *
the definitive socket is either made on the spot or the by otto bock based on the sent in modified test-socket. the final prosthesis is build including the cosmetic parts.

finished definitive prosthesis
the patient tests the finished definitive prosthesis — only some subtle details are modified and the patient can now walk home.

* note: experienced patients might directly jump from testing to the finished prosthesis

starting app, creating patient, check 3d-model, building

today hans-dieter takes his time to »serve« bernhard, an above-the-knee-patient he has known for almost 8 years now, by software – namely tf-design. normally hans-dieter tries to avoid using the software, and instead just faxes the data to otto bock.
this time bernhard not only needs a new socket, but the whole prosthesis. and he has already asked hans-dieter about the possibilities of a special prosthesis for his knee: c-leg.
the new version of tf-design will allow hans-dieter and bernhard to build several alternative prostheses in order to compare their advantages, prices and suitabilities for bernhard and to order all necessary parts, that is all the parts that are needed and not in stock at hans-dieter's branch.

bernhard takes off his prosthesis. with several special tools hans-dieter measures the length and circumference of bernhard's right femoral very close to the nether regions, as well as some other measurements like weight etc — altogether about 7. he notes each data in the dimension sheet his company has printed for this type of amputation. the whole process takes about 15 minutes.

hans-dieter also takes some digital pictures both of bernhard's femoral and his prosthesis, especially the worn-out socket. he will need them as documentation in his argumentation with the health insurance.

hans-dieter starts tf-design and enters the data. he wants to be really sure and visually validates the changes in the 3d-model. as hans-dieter does not feel too comfortable in the 3d world of the computer he only visually checks the 3d-model at a few different angles — cool, everything looks reasonable.

now hans-dieter begins to assemble the prosthesis in the computer. bernhard watches him, intrigued. they start with a c-leg for the knee, then the foot and the combining parts. tf-design has a huge database with all the otto bock parts available and knows which parts are fitting. based on the selections hans-dieter has made so far it proposes the missing components. although they are quite satisfied with the suggested outcome they try a second alternative, this time beginning with a c-walk for the foot. tf-design warns that there is not enough height for this component, so they opt for an equally flexible SpringLite.

both alternatives would work out nicely, but bernhard prefers the first one, because of the c-leg. hans-dieter orders only those part he does not have in stock.

he lets the software generate a report, attaches the pictures he took and prints it out for the gesellen, who will build the prosthesis once all parts are delivered.

Figure 17: Example of persona-based scenarios (right) with accompanying background informations (left) for fuse.

tical about the extent of our involvement, which in their opinion was way too deep. All previous versions of the applications were designed by the engineers. They were in constant contact with some lead users and had a very deep knowledge of the subject matter. Only with the scenarios we were able to encourage the engineers to supply their knowledge gained over the last years in developing the previous versions. Now the scenarios were an artifact on which they could reflect on and give us valuable input. They criticized some details, clarified others, and agreed with many.

With the scenarios we triggered also a discussion about the setting of the users, and their daily routines. This led to new ideas and many inputs both from product managers and engineers.

The scenarios provided the engineers a head start to focus early in the process on the difficult things, such as direct 3D manipulation and a flexible component selection. This again has advantages and disadvantages. The major advantage is to bring in the engineers early in the project. This gives them a stake early on and avoids typical situations on plan-driven projects in which the engineers often have the feeling that they just have to execute what the product management and design comes up with.
A disadvantage could be that the engineers kill too many concepts too early, arguing with technical impossibilities. Although this has happened sometimes, we usually were able to convince the engineers that they

should give it a try, and in some cases we even could help with building functional prototypes, which demonstrated that it was indeed technically possible (cf. chapter 11.3, p 87). In my experience the advantages of getting input and new ideas from engineers early on outnumber the disadvantages by far.

As mentioned in the chapter *Process* (p 32), on fuse the client did not follow a rigorous process, and therefore the follow-ups to the scenarios have taken place in an informal way. This was different with the Audi project: here we followed the Rational Unified Process. I will describe my experience with scenarios within this setting below.

Scenarios and use cases within the Audi project

One of the goals for the Audi website was to start selling cars via the internet within the next five years. This vision was crafted by the strategic department of Audi together with the strategists of Razorfish.

After doing interviews with potential car buyers and dealers at car dealerships we had gained a better understanding of the buying process and we could come up with a set of personas and different scenarios to meet the users' needs. One of those scenarios was *Setting up a test drive*. Although the vision was to sell cars via the internet, we still believed that any private buyer would like to test drive a new car prior purchase. The scenario was straight forward: users should be able to easily book a test drive via the Audi website and get a confirmation of the appointment within two days. To fulfill this scenario not only the experience with the website had to be excellent, but also the service of all participating parties – including the dealer and customer service.

Shortly after we had finished the first set of scenarios we (in this case information architects) sat together with the technical business analyst, who belonged to the engineering department, and walked through every single scenario. Together we came up with first rough flows of how the scenario could work and which parties were involved (see figure 18). With this transformation of knowledge also the name and the ownership of the artifact changed: from now on scenarios were called use cases – high level use cases in this stage, to be specific. The owner of the artifact was the technical business analyst.

Those use cases formed the basis of the technical architecture later on. Additionally the use cases showed that preparations within Audi had to be made to satisfy the high standards of Audi, which the customers were used to. As not only Audi itself was included, but also the dealers. Even it is not shown in the flow in figure 18 discussions started, such as: But what happens if the dealer does not respond within two days? Should the Audi call center respond? Etc.

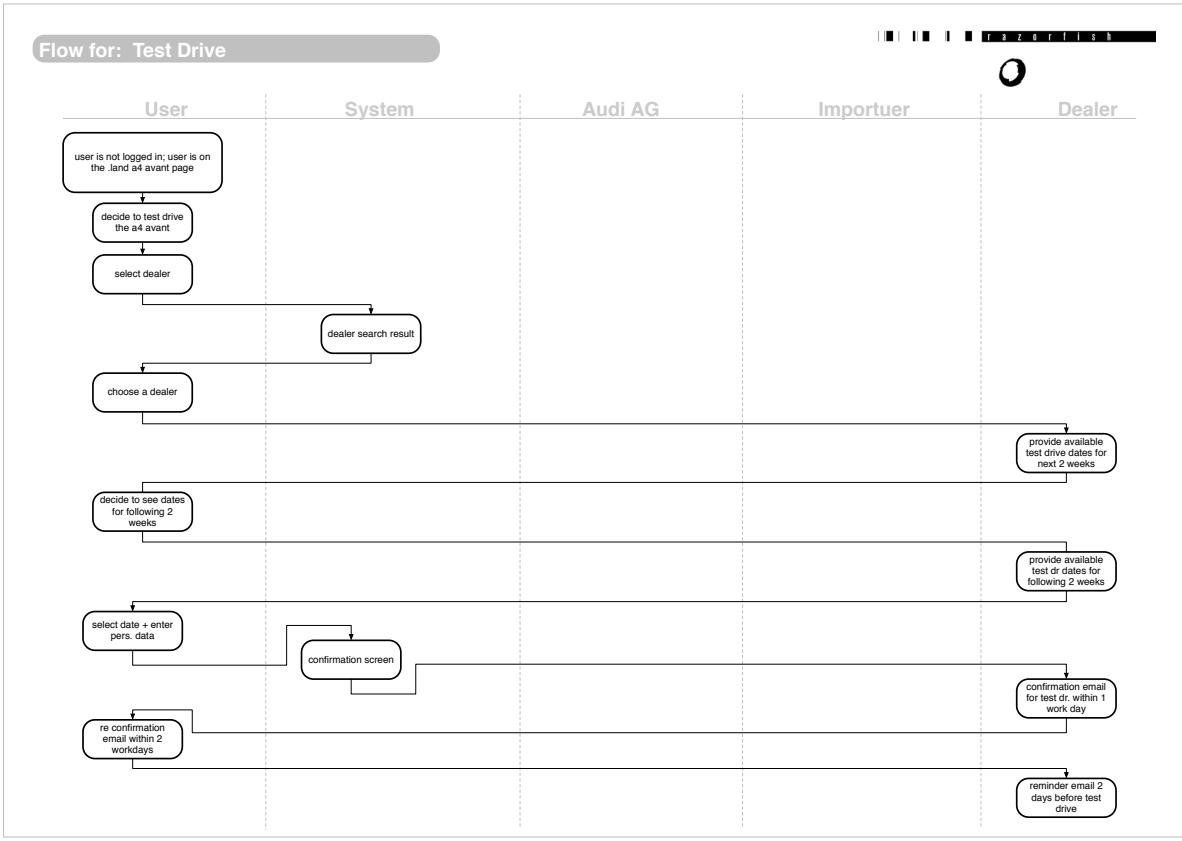


Figure 18: High level use case for arranging a test drive.

With this approach we were able to involve many different departments in the project, long before we had started with any information architecture, interaction design or visual concept for the website.

Scenarios and user stories in the follow-up project of TempRanger

In the follow-up project of TempRanger, which is much larger in scope, we again did an initial explore phase (comparable to up-front design, p 36) to draw a vision of the product for the whole team – and even the whole company. The outcome was presented with presentation sketches (p 93), which followed the story line of typical usage scenarios for the different user groups. By presenting how the vision could actually both look like and function like we enabled a discussion process within the company, which provided us further input for concepts and scenarios.

Later on those scenarios built the basis for the user stories. Together with the product manager we roughly worked out those user stories which should be implemented in the coming iterations. At least one iteration ahead of implementation we then further detail those user stories by preparing the relevant specifications (p 103). Because of the close collaboration the specifications are not as thoroughly described as in plan-driven environments. Nevertheless we cover the flows in detail, describe roughly the functions and prepare the graphical elements and icons.

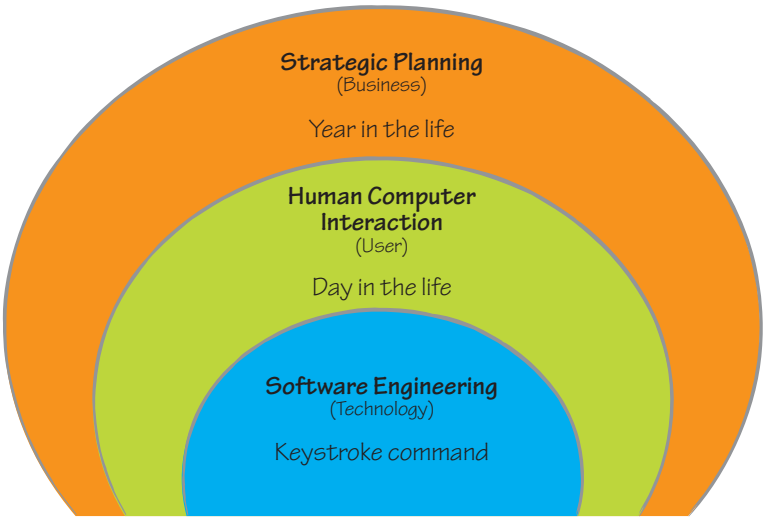


Figure 19: The nested structure of scenarios used in different communities, with the added mappings to the perspectives: Strategic planning – business; human computer interaction – design; software engineering – technology (Go & Carroll 2004).

Working out the user stories on the basis of scenarios has the advantage, of:

- Providing product management with a real world context for deciding which user story should be implemented next.
- Giving the engineers an indication of where the user stories belong in the bigger picture of the product.

Furthermore, by preparing the user story one iteration ahead in more detail, the engineers could focus on the technical aspects, without losing time by either doing the design themselves or waiting for the designers who are working in the same iteration on the same user story.

Scenarios as a boundary object

Go & Carroll (2004) try to build up a common language for the different team members for scenario-based design for software development. The scenarios span different time frames within different communities: year-in-the-life scenario in strategic planing, day-in-the-life scenario in human computer interaction, and moment-to-moment scenario in requirements engineering and object-oriented analysis/design (Go & Carroll (2004) summarize the latter two into software engineering). In addition to the different life span of the scenarios they also deal with a different *degree of tangibility of the target content of the scenario* (Go & Carroll 2004). Software engineering focuses on real world objects, human computer interaction additionally deals with user tasks, and strategic planing covers future plans of organizations. The layered structure over the life span and degree of tangibility is shown in figure 19.

Go & Carroll (2004) conclude that scenarios form a common language for design (figure 20) and foster design thinking:

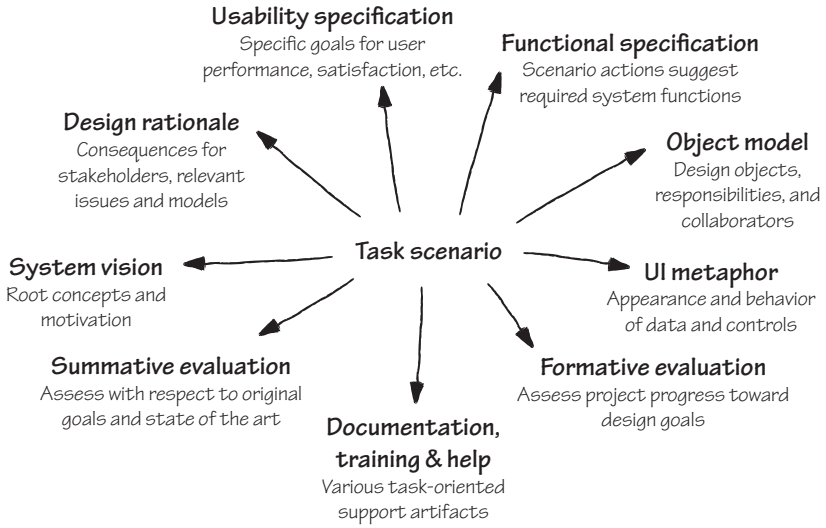


Figure 20: Common language of scenarios for design (Go & Carroll 2004).

> *Scenario-based techniques contextualize and concretize design thinking about people and technologies. […] These different applications of scenarios emphasize different viewpoints and different resolutions of detail, and they address different purposes. But the vocabulary of concrete narratives is accessible to and sharable by diverse stakeholders in a design project: planners and managers, requirements engineers, software developers, customer representatives, human-computer interaction designers, and the users themselves. In this sense, scenarios provide a common language for design.* — Go & Carroll 2004

Our experience with scenarios are in line with the findings of Go & Carroll (2004) and show that scenarios meet the characteristics of effective boundary objects as described in chapter 6 (p 19):

- They have a shared syntax, any form of scenario (e.g. storyboard, use case) can be translated in a plain story
- They allow for learning the different specifics of the scenarios used in the different disciplines
- They support a process and understanding to create a cross-functional scenario. Each team member is able to provide input to the scenarios.
- Scenarios are built in many iterations over the course of a project.

11.3 Prototypes

The term *prototype* is highly stressed in the field of software development. It is used by almost all disciplines involved. Prototyping is a method to quickly and cost-effectively allow for discussion on open points regarding

the (internal and external) design of the final product. Prototypes are an excellent communication tool between the different communities involved in a project, such as users, clients, interaction designers and engineers. To say it with Winograd's (1995) words: *It allows a kind of »reflective conversation« with the materials.*

Prototypes for external design

Rosenberg concludes that over the last 20 years not much has changed in the area of prototypes for external design. The only exception being that the digital products have become more complex and therefore it is not practical to build a complete prototype, which in my experience is rarely ever done, anyway. The other change he mentions is the trend towards agile approaches, in which the prototype becomes the product (Rosenberg 2006). As we have seen in the chapter *Process* (p 32), this is only partly true, because also in agile environments small prototypes for detail issues are very valuable.

Hence the benefits which were already described in the first Handbook of Human Computer Interaction (Helander 1998; the reference is to the 2nd edition, Rosenberg (2006) refers to the first edition) still apply (Rosenberg 2006):

- *provides a means for testing product specific questions that cannot be answered by generic research or guidelines*
- *provides tangible means of evaluating a given UI concept*
- *provides a common reference point for all members of the design team [designers and engineers], users, and marketing*
- *allows the solicitation of meaningful feedback from users*
- *improves the quality and completeness of a product's functional specifications*
- *increases the probability that the product will perform as expected*
- *substantially reduces the total development cost for a product*

But prototypes do not only have advantages, there are also some drawbacks (Rosenberg 2006):

- *limitations and constraints that apply to the real product can be ignored*
- *a prototype can be oversold, creating false expectations*
- *the prototyping process may be difficult to manage and control*

There are many different types of prototypes and several tools and methods for prototyping. They range from paper prototypes to highly functional ones, which mimic the intended product behavior in great detail. This spectrum is usually referred to as low fidelity and high fidelity prototypes respectively. Recently critique on those terms arose (Gedenryd 1998, p 175, Buxton 2007, p 295). Both argue that instead of using terms which reflect how different they are from the final product, terms should be used which reflect their strength. Gedenryd (1998, p 175) proposes *high relevance* or *highly useful*, because those terms reflect more their cognitive purpose.

Arnowitz et al. (2006) give an excellent overview of the art of prototyping, including its history, methods used and tools.

Prototyping is one of the skills every interaction designer should possess (cf. chapter 10.3, p 52):

> *Building prototypes remains the lingua franca of all design professions.* — Rosenberg 2006

For us prototypes are both a tool to run usability tests and an inquiring material (chapter 5.3, p 17; Purgathofer 2003, p 290) to explore the possible interactions. Just for running usability tests, paper prototypes would be sufficient, they allow to discover almost as many usability problems as interactive prototypes (Virzi et al. 1996). But for using them as inquiring material we also need interactive prototypes. In my opinion an interaction designer has to have the knowledge to present his ideas and concepts in an interactive form. Of course in addition to having the skill of building paper prototypes. I think many approaches and also the knowledge of when to use which should be in the skill set of every interaction designer.

Whatever tool one uses does not matter as long as it is interactive enough to show transitions, or animations if this is needed to communicate the user experience. But interaction designers do not need to build sophisticated prototypes. This is comparable to industrial design. Here, every product designer I know is able to build small haptic prototypes. For scale or weight models they usually commission model builders. But with this approach the inquiring process gets lost or stretched over a longer time span.

Regrettably no good tool for sketching or prototyping interactions exists. For the tools we currently use, such as Adobe Flash, some programming knowledge is needed to be able to build the intended concepts. Hence we are still calling for a good sketching and prototyping environment for interactions. Recently Löwgren & Purgathofer (2006) started a promising website to collect best practices and examples on interaction sketching.

Digression: Sketches

I would like to add this digression on sketches, because in my experience sketching is one of the best methods to explore and tinker with different ideas and concepts. Before I provide my reasons for not including sketches as a boundary object I will summarize the differences of sketches to prototypes.

Both prototypes and sketches are *instantiations of the design concept* (Buxton 2007, p 139). But they serve different purposes.

Sketches, the artifacts themselves, are just a by-product of sketching, as an activity (Buxton 2007, p 118). Sketching is a way to explore many different ideas as cheap and as fast as possible. The sketch itself communicates with its representation: *criticize me, I am not finished, give me feedback*.

> *I am disposable, so don't worry about telling me what you really think, especially since I am not sure about this myself.*
> — Buxton 2007, p 106

Even if a designer worked days on a sketch. First, and foremost sketches are a way to communicate with oneself. In a conversation about sketching I had with Christoph Pauschitz, one of my partners at GP designpartners, he said: *We, as designers, sketch, because we can only handle two or three details at once. The sketch is the extension of our brain.*

Buxton summarizes the relevant attributes of a sketch as follows (Buxton 2007, p 111–2):

> *Sketches are:*
> - ***Quick:*** *A sketch is quick to make, or at least gives that impression.*
> - ***Timely:*** *A sketch can be provided when needed.*
> - ***Inexpensive:*** *A sketch is cheap. […]*
> - ***Disposable:*** *If you can't afford to throw it away when done, it is probably not a sketch. The investment with a sketch is in the concept, not the execution. […] this doesn't mean that they have no value […]*
> - ***Plentiful:*** *Sketches tend not to exist in isolation. [Their] relevance is generally in the context of a collection or series […]*
> - ***Clear vocabulary:*** *The style […] follows certain conventions […] The way that lines extend through endpoints is an example.*
> - ***Distinct gesture:*** *There is a fluidity to sketches that gives them a sense of openness and freedom. They are not tight and precise […]*

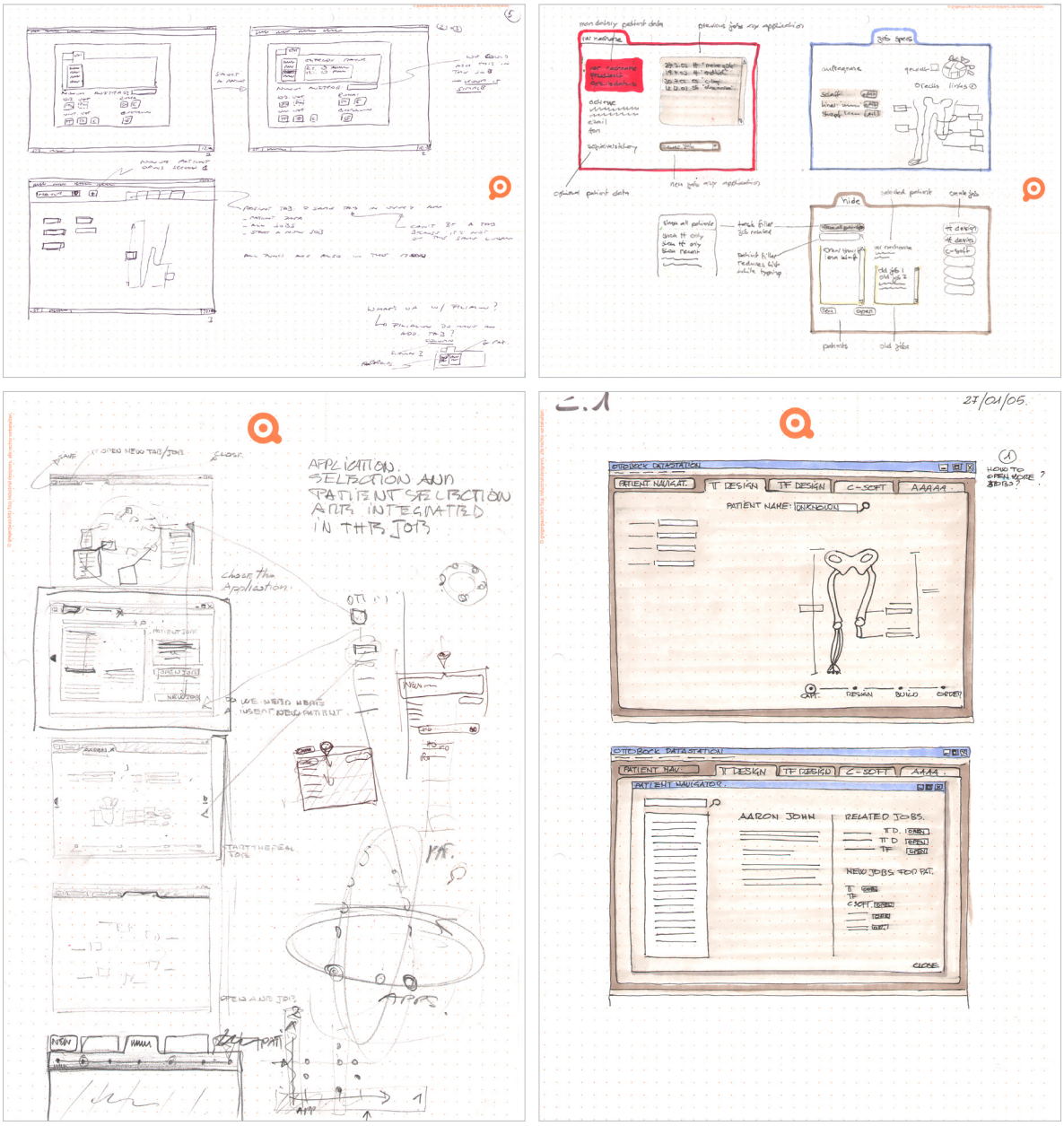


Figure 21: Sketches for the project fuse from different team members.

- ***Minimal detail:*** *Include only what is required to render the intended purpose or concept. […] Going beyond »good enough« is a negative, not a positive.*
- ***Appropriate degree of refinement:*** *[…] As Lawson (1997) expresses it, »… it seems helpful if the drawing suggest only a level of precession which corresponds to the level of certainty in the designer's mind at the time.«*
- ***Suggest and explore rather than confirm:*** *[…] sketchs don't »tell«, they »suggest«.*

› ***Ambiguity:*** *Sketches are intentionally ambiguous, and much of their value derives from their being able to be interpreted in different ways, […]*

Besides the communication with oneself, sketches also ask for input from the other team members. My sketches often have short comments about the driver of the idea, or the way something should behave (see figure 21 for different examples of sketches). Those notes also help in our sparring meetings, usually one on one, occasionally more, in which we reflect on each others sketches. The one who describes his way of seeing the situation often uses scenarios to provide the context he imagined while he did the sketch. Usually we only discuss a fraction of the sketches we did, but we do show all. Sometimes another designer gets inspired by a sketch, of someone else. Even if this sketch might not be of any importance to him anymore at the time of the meeting, he just needed it to reach some point, at which he wanted input. It is important to note that the creation and reading of sketches is a skill which one has to learn: *It takes the same kind of learning to acquire the skills to converse fluently with a sketch as it takes to learn to speak in any other foreign language.* (Buxton 2007, p 118).

Choosing some of the sketches to discuss them with other team members is the first action to single out certain concepts. Later on many more of these decisions are made and the ideas – and with them the concepts – get more concrete. With this transition more sophisticated prototypes are created. The ideas close in on certain concepts. Buxton shows this in the dynamics of the design funnel as shown in figure 22 and talks about *things, [which] are converging within the design funnel* (Buxton 2007, p 139). The production of a prototype requires additional time and money. Hence there are fewer prototypes than sketches.

Buxton summarizes the continuum from a sketch to a prototype nicely by their different characteristics. The arrows symbolize that it is a continuum and not an either/or proposition (Buxton 2007, p 140):

Sketch		***Prototype***
evocative	→	*didactic*
suggest	→	*describe*
explore	→	*refine*
question	→	*answer*
propose	→	*test*
provoke	→	*resolve*
tentative	→	*specific*
noncommittal	→	*depiction*

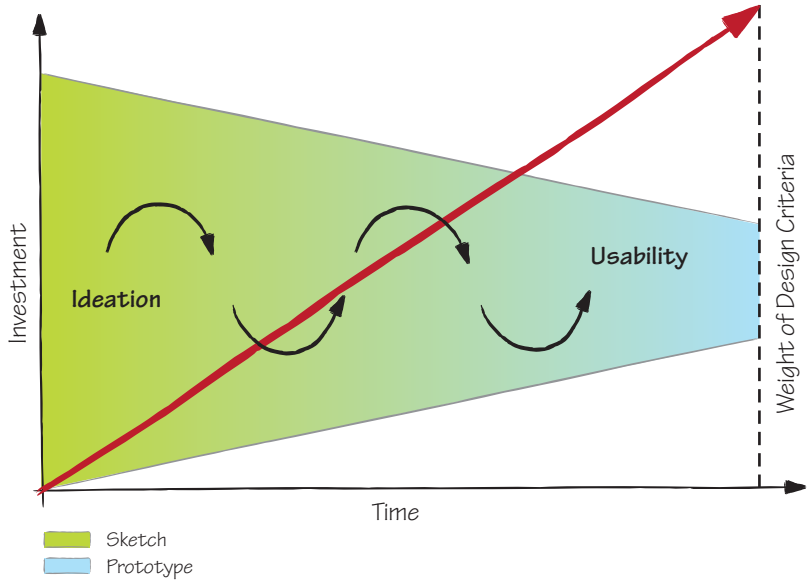


Figure 22: The dynamics of the design funnel: In the ideation stage many sketches are produced. To the right of the funnel testing gets more important and as such the ideas have to get more concrete and the need for (more expensive) prototypes rises (this is shown with the green and blue shading). The red arrow indicates the rising overall investment in the process over the progress. With the increasing investment the weight of the criteria for evaluating design decisions should also rise – *you don't manage ideation the same way, or with the same rigor, as usability.*
The circular arrows show that users are involved throughout the process and not just for usability testing. (Buxton 2007, p 139)

Why is a sketch not a boundary object?

As mentioned above the sketch itself is just a by-product to the activity of sketching. So the sketch alone does only communicate a certain degree of information. Furthermore one has to have the skills to read sketches, therefore it cannot be used equally by all team members. The activity of sketching is very immanent to design thinking and doing for the sake of knowing. This is a new approach especially to engineers, who usually work in a deductive style.

We often use annotated sketches (figure 23) in client and team presentations within the explore phase. We call those sketches *scribbles*. Scribbles are comparable to what Lawson (1997) named presentation sketches. Despite being annotated in much more detail than during normal sketching, they only really work if someone follows the presentation itself, in which the design rationale is presented verbally (cf. chapter 10.4, p 55).

Before preparing a presentation usually internal design team meetings take place in which the individual designers make a first selection. In the client meeting itself a further selection process happen. Therefore I would say that those presentations are much rather storyboards or very early prototypes, which are included in the artifacts as boundary objects.

Sketch a project – a digression within the digression

Reading Gedenryd's (1998) *How Designers Work* and Buxton's (2007) *Sketching User Experiences: Getting the Design Right and the Right Design* triggered the following idea: What tools or instruments are needed to enable sketching in setting up a project and bringing design thinking to the team members? After researching our own projects and case studies from different viewpoints, I needed:

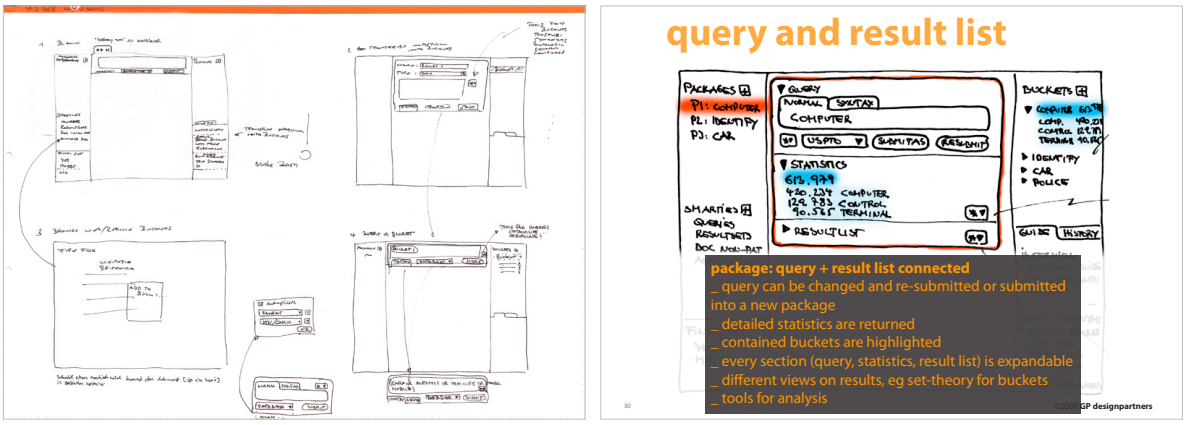


Figure 23: Examples of sketches on the left vs annotated sketches (scribbles) in a presentation on the right (these examples are taken from the follow-up project of TempRanger).

- A way to kick off the sketching process, because especially project managers and team members are not used to sketching with pen and paper.
- A way to summarize my findings.

I very much believe in interactive project setup meetings with the whole team included for project planning. Of course this only works up to a certain team size. For larger teams this can be done within the core team and within the individual teams respectively (cf. chapter 10.7, p 68). This is often done only with post it-notes and a timeline and a lively discussion of the team. But this lively discussion often does not kick off.

Additionally I discovered the method cards from IDEO, and fell in love with them. Even when knowing about almost all methods mentioned, they provide a perfect tool for inspiration and as a starting point for discussions with my fellow designers.

Those were the important drivers for the creation of the catalyst kit (p 121). By which I hope to bring sketching into project management and project setup. Project management is foremost about communication and the understanding of all different needs of the team members and the client, not only about allocating tasks and tracking deliverables.

Prototypes for internal design

Prototypes are not specific to external design, they are also used by engineers for internal design. Here evolutionary prototypes are more common. We experienced that engineers are often puzzled when we tell them, that:

- Our prototypes will be usually thrown away.
- We purposely build our prototypes in a language and with tools different from those of the final product.

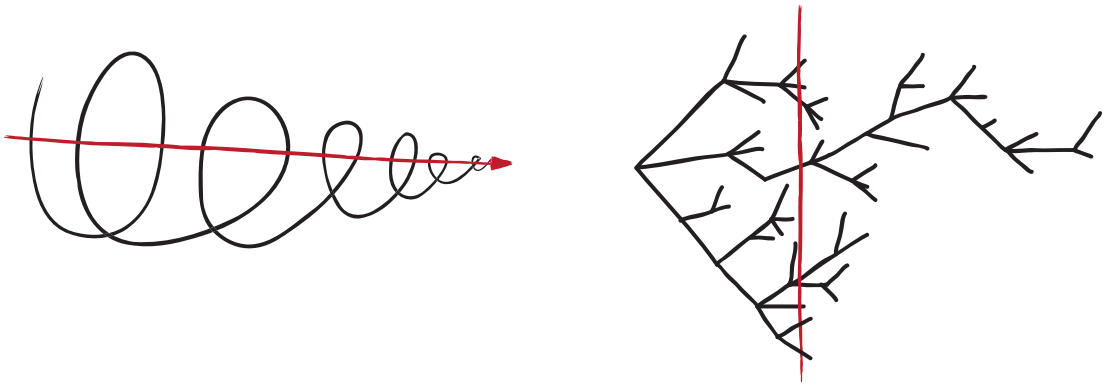


Figure 24: The left drawing shows the usual approach in engineering. Each prototype is a refinement of the previous one. The right drawing shows explorative elements of sketching and prototyping, which is typically used within the design discipline. Many different alternatives are considered. (Buxton 2007, p 388)

Buxton (2007) shows the difference between those two approaches to prototyping in figure 24.

The important thing about prototypes is that each discipline can learn from their own and from the prototypes of the other discipline. Technical prototypes show constraints, but also provide a source for new ideas. Therefore it is important that an active exchange of the findings can take place.

Our experience with prototypes

In projects we usually run through a number of different prototypes, they range from paper prototypes (early on to explore different metaphors) to functional prototypes (for documentation and sometimes also as showcase for marketing). With most of those prototypes usability tests are conducted, either just within the design team or with *real* users. We do not just build prototypes to be able to run usability tests, we see them more as a tool for exploring and getting a feeling and understanding of the actual product.

In our experience the activity of prototyping is similar to sketching in regards of the reflective dimension. We use prototypes to communicate with ourselves, to find out the behavior of the interactions in detail. Only by building those prototypes we were able to find out about certain flows. Sketching with pen and paper is sometimes too static to discover certain behaviors and flows of a software. When building the prototype we run into many details which we did not think of before. Those issues might be solved directly in the prototype or we use sketching to explore the different possibilities. If necessary, we sometimes try out different approaches with prototypes to better understand the problem and our solutions. But we build far less prototypes than we draft sketches. We only do them if we have the feeling we are not able to find out with sketches.

As the next chapter *Specifications* (p 103) shows even during prototyping we still do not catch all details. This is a good thing. With every

method we figure out the level of details which is currently needed. Sketching does not stop until we deliver the project.

Different kinds of prototypes in the project fuse – always with the right fidelity

Early in the project, as early as writing the first scenarios, we started with sketches as shown in figure 21. Because of the wide range of different applications we had to bring onto one platform, we had to deal with two different kinds of problems:

- Finding a metaphor for bringing all applications together.
- Finding concepts for many detailed problems – as for example: 3D-manipulation, component selection for a prosthesis, setting marks on photos, capturing 3D-data, or adjusting a technical device while in operation on a patient.

I will describe my experience with the prototypes for the component selection for a prosthesis.

In the previous version of the software the component selection was handled by a wizard-like process. The selection process is very complex, because it has to fulfill many constraints, such as functionality of the knee joint and foot, compatibility of different parts, and fitting within a certain height. The wizard-like process was driven by two things:

- Microsoft's utilization of wizards for almost everything. To the engineers wizard were almost the answer to everything. We experience currently a similar approach on a different project which will be implemented within the eclipse framework. Here, also everything is handled with wizards. I do not know where this belief in wizards stems from.
- The complexity of the interdependencies of the components. With a wizard the engineers were able to funnel the users through a question and answer marathon to narrow down the possibilities for the next step.

Zoo Mix Max
is a simple dice rolling game for children. The objective is to build correct animals from the different animal parts (head, neck, body, feet) which are picked up by the dice.

The problem with the wizard-like approach was that the users did not like it. Because it did not meet the selection process they were used to. Usually the medical technicians start with a certain knee joint or a foot. This selection is often influenced by the patient, who has preferences towards a certain type, and by the social insurance companies, which only allow for a certain budget.

During the first meetings with the client and engineers we discussed the problems with current component selection – and a game we played

Figure 25: Prototyping with Zoo Mix Max. A perfect tool for a quick and inexpensive exploration of our idea of an open selection process.

as children came into my mind: *Zoo Mix Max*. The game became our first prototype (figure 25). With Zoo Mix Max we were able to quickly explore our idea of an open selection process. The selection of any component at any time was possible. By simply simulating the mouse click with a finger tap and then pushing the according row in such a position that the selected part becomes a part of the »prosthesis« we gained a better understanding of how the idea could work.

We developed this approach further and built a simple prototype (figure 26). In our first presentation (after the explore phase) we showed our findings of the user research together with the personas and scenarios and first concepts as scribbles and early prototypes. For the component selection we performed a live prototyping session with the game and presented the more elaborated prototype.

The prototype shows that the selection process could be started with any component, and that any component could be changed at any time. With each selection the new possibilities are computed and shown (suggested ones, possible ones, and impossible ones). We further more imagined that the system should provide a reason why certain components are impossible or not practical to use. We even allowed that the medical technicians were able to select impractical components (shown in step 5) – they are the expert and thus know what they do. Of course, it then could happen that some already selected parts become impractical, but others would be suggested to solve this situation. We believed in a system which supports exploring of different options.

This early prototype was an excellent way to present our concept. It allowed all members involved to give input and feedback. Marketing and

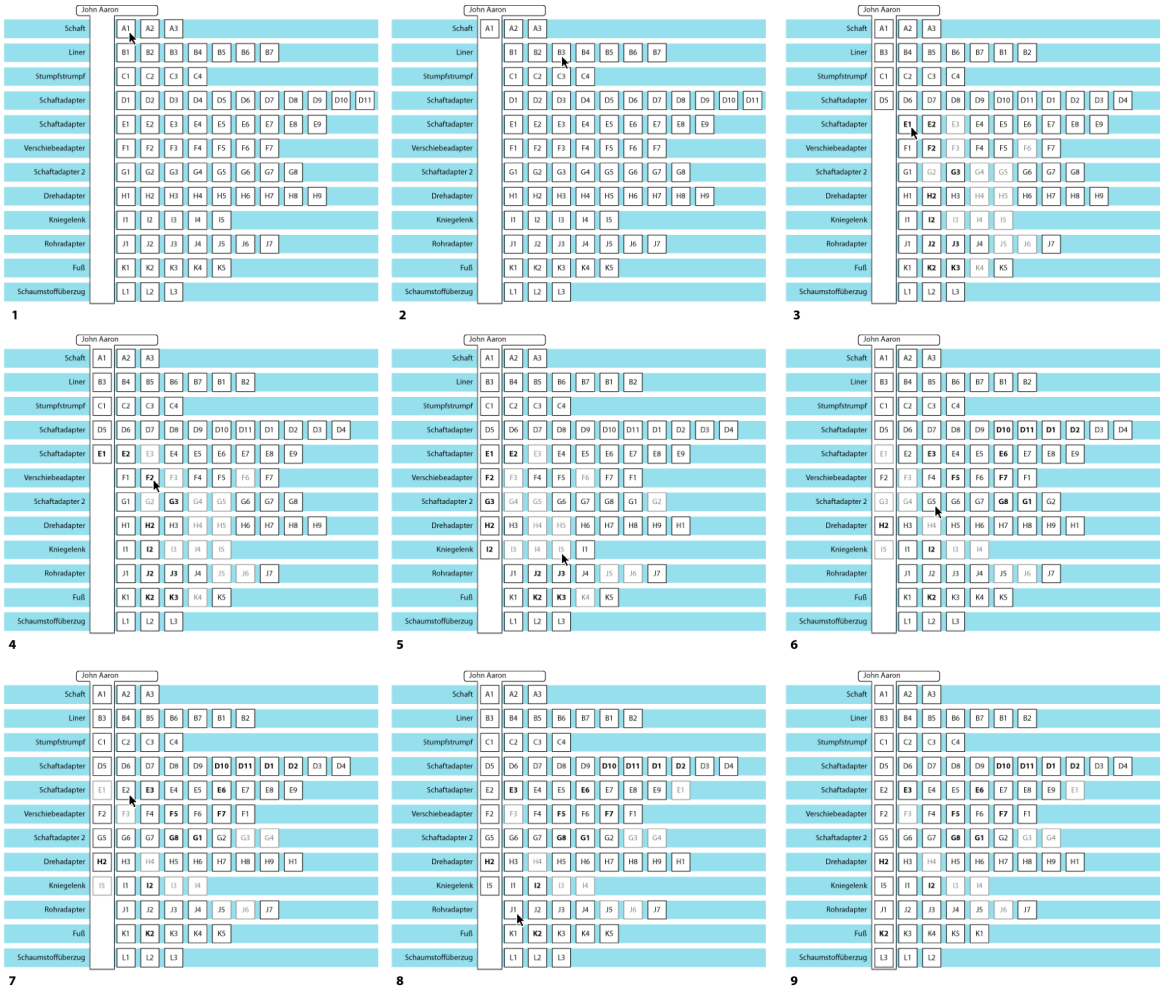


Figure 26: Screens of an early prototype for the component selection: Each selection indicates the new options (suggested, possible, and impossible). Every component can be selected by the user, even impossible ones. The medical technicians are the experts and thus know what they do. The system provides guidance but not restrictions. The prototype was built with a presentation software, namely Apple Keynote.

product management loved our proposal, because with this approach they were able to bring new components into the market. Often the medical technicians stick with their standard components, even if there are already better ones on the market. They do not have time to monitor the market all the time. The engineers in contrast were – to phrase it positively – very skeptical. The concerns could be grouped within three categories:

› Technical issues: Those issues were very essential. The product catalog contained about 20.000 items, stored in some sort of relational database. They did not know if it was feasible, regarding the performance, to come up with suggestions and possibilities with each click. But at least we persuaded them to give it a try. We also asked them if they could provide us with the data (we already had the catalogues in printed form) and the entity-relationship models for the database (cf. chapter 11.5, p 118).

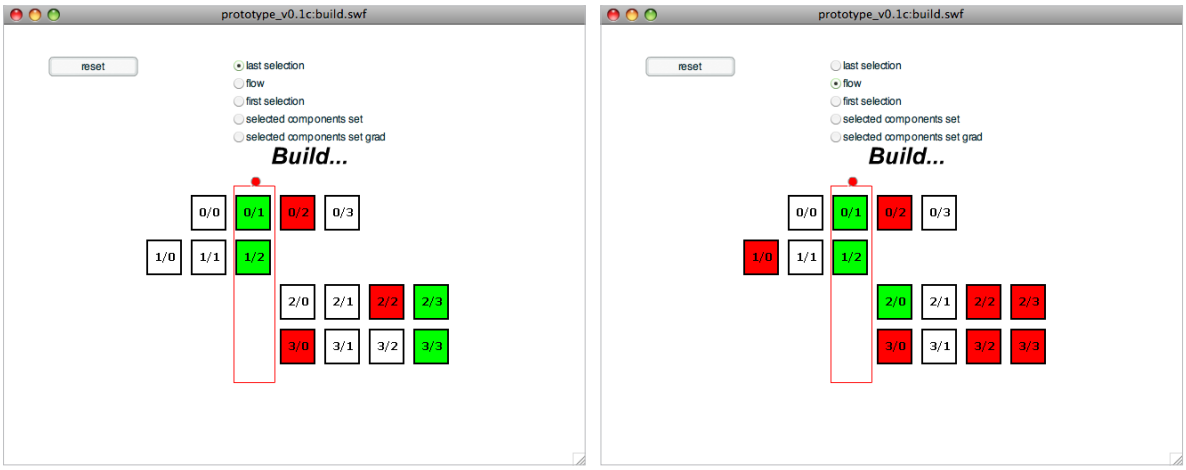


Figure 27: A clickable prototype to test the different approaches for calculating the suggestions and impossibilities for the component selection. Only with this prototype we were able to decide for a calculation approach.

- Organizational issues: They brought up another important issue: who will decide which components are suggested? Because of their long lasting involvement over many years in the project they knew about liability issues and the decision culture of the client.
- Users: They believed that the users need more guidance with the selection. This believe might be triggered by the technical issues, but also in defense of their current solution. Furthermore, they still did not like that we got such a huge stake in the project. In their understanding we were only here to make the interface pretty for the functions they came and come up with. In our experience such issues are often covered as user issues, because they are the least tangible ones. A lot can be influenced by lobbying and political leverage, covered as users issues. Personas are a good tool to tackle such attempts early on.

At this stage of the project we also did not know how our approach would work in detail. But with the presentation of an early prototype we were able to get viable feedback early on.

Besides paper prototypes for the overall concept, we built a first interactive prototype for the selection process. The purpose of this prototype was to gain a better understanding of the different strategies of how the suggestions could be calculated (see figure 27). Only by building this prototype we were able to explore the different strategies. The process of doing it triggered new approaches.

To be completely honest, this was also a test or exploration of which prototyping tool we should use.

Based on the findings of the paper prototypes for the overall concept (which we usability tested internally) and on the functional prototype for the component selection we built the first version of an overall prototype to run first usability test with friendly users. Figure 28 shows the prototype, which included the general workflow through a job: creation of

Prototyping tool
We decided for Adobe Flash. Flash gave us the flexibility to prototype the 3D-environments and allowed us to introduce new controls easier – compared to tools such as Microsoft Visual Basic, which would have given us the advantage of not needing to fake the standard controls at the cost of only having standard controls.

Figure 28: Sample screens from the first version of the interactive overall prototype.

new patients, selecting and setting up different jobs, input of dimensions, visualization and manipulation of the 3D model, component selection and ordering.

In general we got good feedback at the usability test, but also discovered many issues – some of which we did not expect at all. The users especially liked the explorative element of the selection process, but were unconvinced of some of the details. Meanwhile our relationship to the engineers deepened, although their technical concerns about the component selection remained. They were also doubtful if our approach still would work for the users if all of the products were included in the selection process – and not only the small fraction we did put into the prototype.

In the next version we addressed the issues gathered in the usability tests and included new parts. Additionally we wanted to demonstrate that the selection process would work with all components included. Luckily already the first version of the prototype was based on an object oriented structure, so it was not too difficult to extend it for the whole range of components. Besides the technical change, we improved the feedback system, added measurement details and a suggestion system (see figure 29).

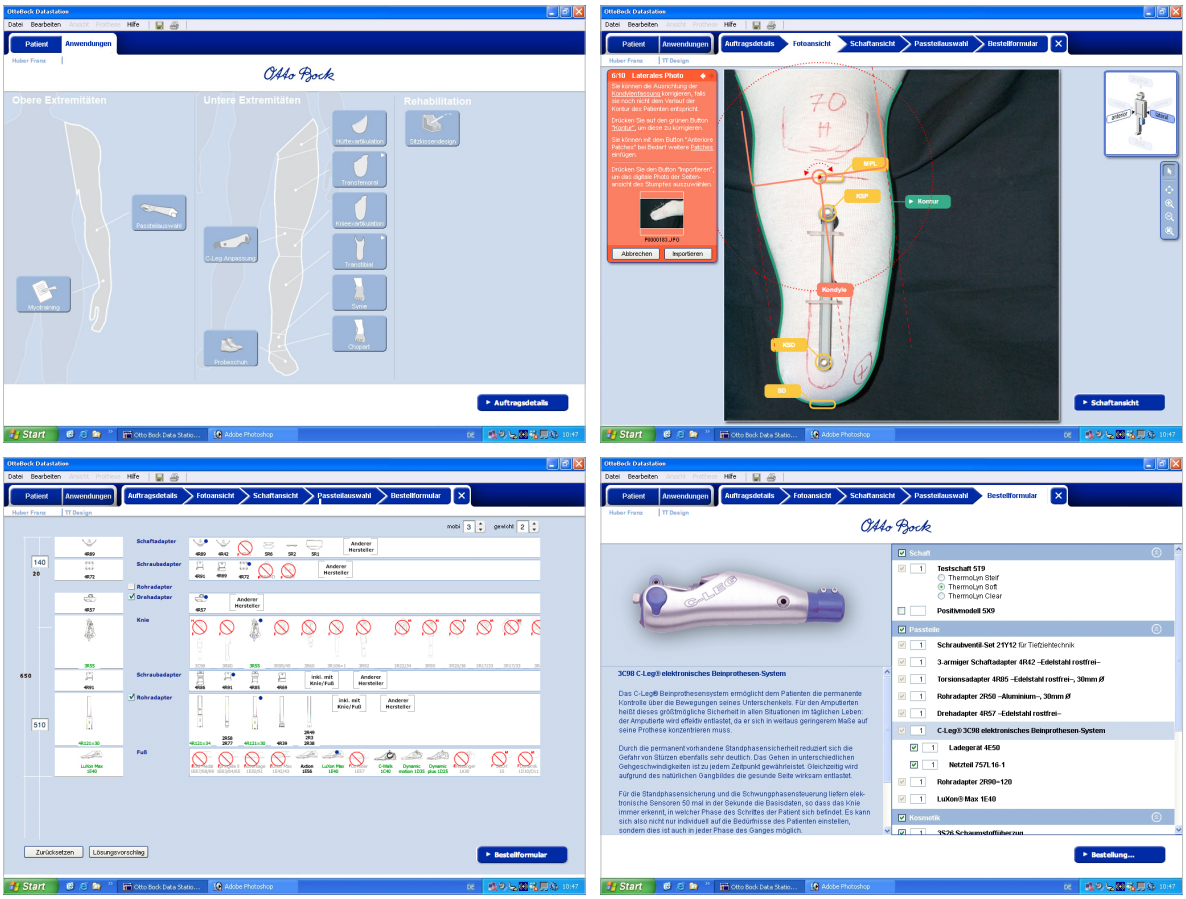


Figure 29: Sample screens from the last version of the interactive overall prototype, which was used for usability tests and acted as a *living specification* for the engineers.

The results from the usability tests were again very positive. The remaining issues were directly tackled in the functional specifications (p 106). With this prototype we finally convinced the engineers. When we, with – at best – rudimentary programming skills were able to implement a solution which not only worked, but also had a good performance, it had to be easy for them to do it even better.

Prototypes within agile environments – TempRanger

TempRanger was developed within an agile environment. When strictly following agile development prototypes should not be necessary. But to integrate user-centered design better in agile environments Miller (2005) and Chamberlain et al. (2006) (cf. chapter 9.2, p 35) describe and suggest the usage of prototypes. However, compared to projects in plan-driven environments we build mostly paper prototypes.

On TempRanger I would not even call them paper prototypes, they are annotated sketches (figure 30 left), as described in the digression on sketches (p 90). From those we jumped directly to actual screen designs and graphical elements in Adobe Photoshop for the upcoming user stories. We, as interaction designers, worked at least one iteration ahead of the engineers. In the following iterations those screens and graphical ele-

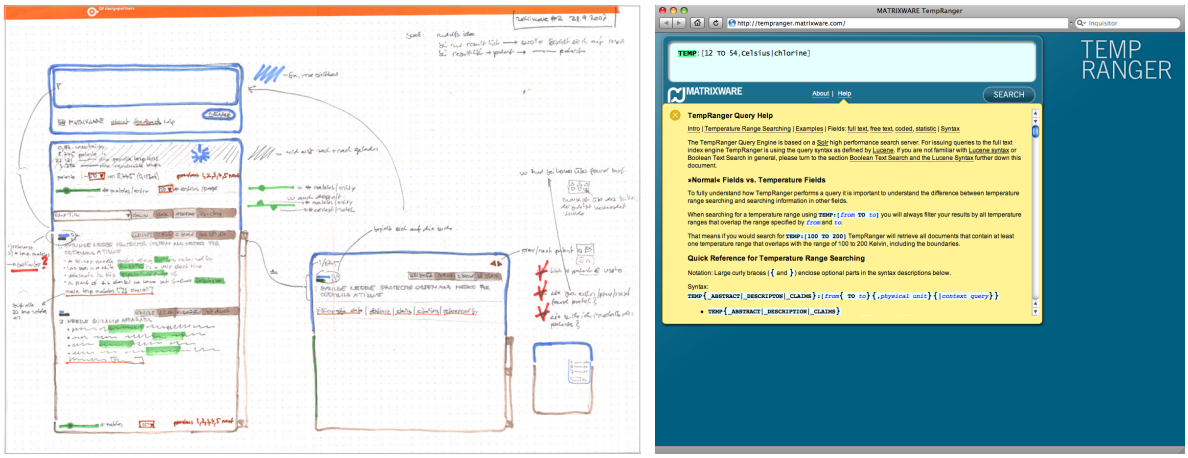


Figure 30: Left: A sketch of TempRanger. Right: A screenshot of the help section opened directly from the error message by the user after an error occurred (sadly, transitions are hard to show on paper).

ments were implemented. During development of certain flows, especially ones with animations in it, we sat together with the engineers (in this case front-end developers) and played around with the animations, transitions and timings. This was done on a branch of the actual product, but without sticking to programming guidelines. It was tinkering in the code to see different behaviors. For me this method can be called prototyping. For example, for error handling and the help system we tried different approaches (figure 30 right). Played around with each one. Gathered input from team members. And then decided on the design. Only after this decision the code was transferred into a production ready state.

Thomas Piribauer, a user experience consultant, mentioned a similar approach in a project in the telecommunication business in a discussion at our local face to face IXDA-meeting (Spangl & Haberfellner 2004) about user-centered design within agile environments. He called this approach *pair design/programming* with reference to pair programming in extreme programming.

Prototypes as boundary objects

Prototypes are used in both disciplines. Because of their different usage – as throw away prototype for explorative purpose in design and as evolutionary prototype, for proof of concept in engineering – it is important to discuss the intentions of the prototypes. In contrast to the other artifacts here even the naming of the artifact is the same in both disciplines. At first sight this might be seen as a positive point, because the term means something to everyone and does not need to be introduced or translated to the other discipline – but at the risk of assuming that everyone has the same understanding.

Especially for this artifact the fifth characteristic of a boundary object, the discussion process in building the boundary object itself, is significant (see chapter 6.4, p 24).

Once mutual understanding of the different types of prototypes used in the project is established they are an excellent tool for discussions and for bringing value to all team members involved. Be it as the process of creating them or as the artifact itself, which transports the intended concept in a more tangible way. In my experience it is hard for clients to get a concise picture on how the software behaves from abstract drawings or static screenshots together with flows (p 107). Additionally the prototypes serve as guidance for the engineers when implementing the product. And of course they are perfect for running usability tests.

To conclude, for us prototypes are a way of communicating with ourselves, with the users, with the client and with the engineers in order to gather important feedback and findings.

11.4 Specifications

I use the broad term *specifications* to cover the most relevant artifacts, which define the interaction design for the software in detail. Within the specifications the findings and decisions are summarized with a focus to support the implementation of the findings – the specifications have to be edited in an implementation friendly way:

> *[Design] will never get built unless it is described at length, with precision and detail, in terms that make sense to the programmers who must build it. It has to be in writing, in exhaustive detail, with supporting evidence and examples. […] The designers need to write, storyboard animate, and sketch their solutions with sufficient completeness and detail that the programmers can treat the solutions like blueprints and actually write code from them.* — Cooper 2004, p 226

Because of budget and time constraints not everything can be described at the same level of detail. It is the interaction designer's responsibility to focus on the most important things (Cooper 2004). Additionally they are accountable to gain an understanding with the engineers which parts need to be captured in what level of detail. The purpose of the specification is also to transfer the process of developing the design to the engineers, so that they are able to decide on some of the details by themselves.

In plan-driven environments the misbelief that everything can be specified in the design phase already is very prevalent. In reality this is never the case. We are happy if we are able to cover about 80 percent of the functionality. The important thing is that the understanding of the depth of the coverage is shared among all parties. This prevents upcoming conflict situations early on. I have witnessed too many situations in which

engineers complained about the lack of specifications of some features. On the other hand, the coverage of only a certain amount of depth should not be taken as an excuse of the designers to be sloppy with the documentation. It is in the interest of the team – and in the responsibility of the designers – to shape the transformation of the design into implementation as smoothly as possible.

The main critique point on specifications is that many decisions are made with only little information available (37signals 2006, Beck et al. 2001). 37signals (2006) bring up that functional specifications *only lead to an illusion of agreement*, because people agree only on written text and everyone interprets it differently. In my opinion this critique should be passed on to the direction of the process. The specifications are just a vehicle within this process. But I agree that in traditional plan-driven environments the customer requirement specifications are defined mostly only in written form and signed off too early in the process. This often happens without the input from users and the involvement of designers. On many of our projects we get customer requirement specifications of diverse quality and are asked to design the interface or write a user interface style guide. In most cases, we are able to convince the client that the customer requirement document is just a starting point for us, and that we will question its content during the process of the design and convert them into specifications, which are focused not only on the customer but also on the users and other stakeholders.

When I talk about specifications in this thesis I relate to the outcome of the complete phase in our process (p 41), which corresponds to the outcome of phase 0 in Buxton's process model (p 34) or the knowledge transferred in the workshop as described in Purgathofer's butterfly model (p 33). More often than not our specifications then become the requirement specifications.

37signals (2006) argue that the interface should be the functional specifications, and that after some sketches the coding should start. In my opinion this is only possible within small teams and with highly skilled team members – which is usually not the case in larger projects in the industry.

Our approach to handle this is to document the interaction design as close as possible to the actual product in the form of pixel accurate screenshots (p 110), flows (p 107) accompanied by functional specifications (p 106) to cover the behavior and functionality, and an interaction design style guide (p 113). Often also the latest prototypes are used by the engineers as a *living specification* for the product. In my opinion the right combination of those artifacts is important, as any single one, when taken alone, cannot transport the richness of the interaction design in detail. But not always are all of those needed. To say it with the words of Cooper:

> *As well as knowing what is good design, the designer must know what is important. The interaction designer must decide which parts of the program must be designed, and which parts can be left to the indigenous solutions of the programmers.*
> — Cooper 2004, p 227

But even more important is that the interaction designers transfer the drivers and the way of thinking to the engineers and the team, so that decisions, on those parts which are not designed in detail, are based on the right fundament.

Before I describe the specification artifacts and our experience with them in more detail I will briefly talk about specifications within agile environments.

Specifications in agile environments

Are specifications in an agile environment obsolete? Following one of the statements in the agile manifesto (Beck et al. 2001): *Working software over comprehensive documentation*, documentation and with it specifications become less important. But this does not mean that no documentation is done at all.

The documentation of functionality in the form of user stories, and the creation for a backlog to collect all of those stories is typical for agile environments. The client and the engineers then agree on which of those stories to implement in the next iteration. As already discussed in chapter 9.2 (p 35), the user stories should rather be called customer stories currently, due to a small degree of user involvement. But some specification is required following the conclusion in the chapter *Process* (p 45) on the integration of users and design into agile environments (with up-front design and/or integrated design one iteration ahead).

In agile environments the development is on a per feature basis, as well as are the specifications. In our experience it was important for both the product and the productivity of the engineers that we, as interaction designers, elaborated and detailed the user stories, and to define them together with product management in such a detail that the engineers could focus on the implementation rather than on the design of the user interface, which they had to prior to our involvement in the project. The engineers started to develop on the basis of very rough and very short user stories (mainly not exceeding two or three lines). Because of the constant collaboration within the team, the specifications are not as formal as in plan-driven environments. We frequently only specify the screen designs and then walk the engineers through the flows, and they take notes. If questions and problems arise during the implementation we clarify them together on the fly.

Functional specifications

The functional specifications describe the functionality and interaction concepts. They integrate all other specifications, such as flows and pixel accurate screens and are based on the general rules defined in the interaction design style guide. Elements are specified in detail in their respective context of use, e.g. sort order in drop down-fields, preselected entry, etc. Furthermore they define the interaction and the resulting consequences for every element and what should happen in case of an error.

The depth of the specification depends on the project. Depending on the size of the team and project, complexity of the project, and process followed, we choose the most appropriate types.

On fuse we did the full range to cover all details for the seven applications. We used an internal *wiki* to collect our decisions throughout the project. The plan was that the internal wiki itself should become the specifications. We first thought that after a bit of cleaning up and polishing we could open it also to the client. Allowing them to directly comment the specifications, while they still were written in detail. This did not work out. Firstly the internal wiki was too unstructured and contained many notes and ideas, inherent to our process of forming the product. Secondly the client was not used to a permanent involvement. Thirdly, we did not set up a good feedback-process. We thought that turning on the commenting feature in the wiki would suffice. This worked technically, but not socially. It was unclear when one should comment on which sections and features.

So in the end we started a new wiki and the internal wiki formed only the basis for the specifications (see figure 31). Once the specifications were finished on our part we asked for a review, followed by a sign off on a per feature basis. An interesting side note: our client still requested the specifications in pdf-format, or differently stated, they wanted an *actual* document, in the traditional sense of being pages-based. Another point to mention is that wikis are barley known within the companies we worked for.

Besides the failed client involvement, using a wiki worked brilliantly. We, three interaction designers, were able to build up the documentation simultaneously and rely on resources from other team members.

In another project (also set in a plan-driven environment) we factored in the findings from fuse and set up a feedback-process. We added a status for each feature (in reality for each page). Interestingly the client did not start commenting until all features were specified. They were so used to follow their general document flow, in favor of the possibility to provide feedback early on.

Nevertheless, I believe that providing options to give feedback at early stages will change the behavior of the team members in a plan-driven en-

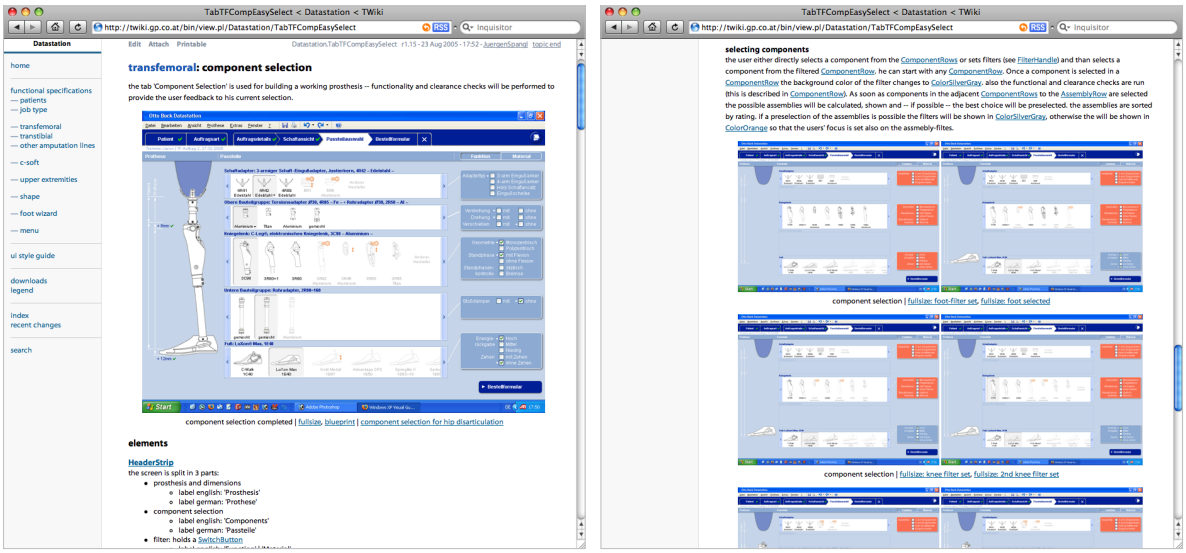


Figure 31: Functional specifications for the component selection.

vironment over time. They will get used to the fact that everyone can get involved throughout the project and not only at certain predefined points. This is also supported by the fact that the client of the project described above set up their own wiki-server and we transferred our specifications onto their environment. Furthermore this is one of the few projects on which we are involved also in the implementation.

Getting constant feedback from the client and engineers works better in agile environments. In the follow-up project to TempRanger all team members are used to daily scrums and biweekly sprint meetings. Everyone is used to wikis and collaborative tools. Here we are specifying the features on a per user story basis within the project-wiki, already set up by the client for their internal use. Because we work off-site we are not involved in every daily scrum. Hence part of the communication runs through comments and answers directly in the wiki.

Flows

One of the hard parts in developing software is to design the transitions:

> *The user's experience is shaped as much (if not more) by the transitions as it is by the states. Therefore they must be equally in the forefront during the design process. Yet, in my experience, this is seldom the case. Attention to the transitions nearly always has lagged far behind.* — Buxton 2007, p 293

Methods such as scenarios and prototyping are a good way to gather a better understanding of the context and to tinker around. Furthermore they allow to quickly show other team members the interaction with the product. Although prototypes are useful to try out, test and even docu-

ment transitions and details of the product, they are too expensive to build and usually fall short of covering all details.

At a certain point in the design process more attention to details and exception cases (such as error handling) has to be given. This usually happens once the overall concept is pretty stable and the technical feasibility is checked. We define those detailed interactions with flows, similar to common process or flow charts. But instead of using the same syntax for all projects or even a formal syntax, we adapt the syntax depending on the needs of the project.

Flows in the project fuse

On fuse we used an abstract syntax for the flows, because the specifications consisted also of a prototype covering many parts of the application in detail. Thus we used the flows mainly for detailed interactions, such as the simple transaction of changing the focus in a list box from one patient to another one (figure 32)

When drawing these flows I am completely immersed in the software and imagine what-if scenarios. With this approach I stumble over exception cases, such as: During user research we found out that for upper extremities products the medical technicians have to run tests to figure out whether the patient is capable of a myo-electrical prosthesis. They were used to just running the tests without entering any patient details, because if the patient is not capable there is no need to save the patient data. Therefore the new software had to be flexible when entering the patient data and not forcing the user to enter it only to run a test and then never to be needed again. But if the patient is apt to a prosthesis the test results need to get stored in a job file, without running the test again. We made this possible, but it caused a more complex handling when selecting a different patient while a job is open. By drawing the flow we were able to nail down each branch and design the dialogs for each case. Instead of just briefly stating the exception, we documented the complete dialog message and button layout directly in the flow. The actual copy is often forgotten in the design, despite being a major part in creating a good user experience (37signals 2006).

A few months before we started documenting, approximately in the middle of the phase shape, we checked with the engineers, whether our flows would serve their needs. With this approach we got valuable feedback at which abstraction level we should develop the flows and where to best document the details for the dialogs. Furthermore we gained additional trust from the engineers that we are capable to actually help them during implementation and not just slow them down – which sadly is still a common thinking among engineers (Tognazzini 2005).

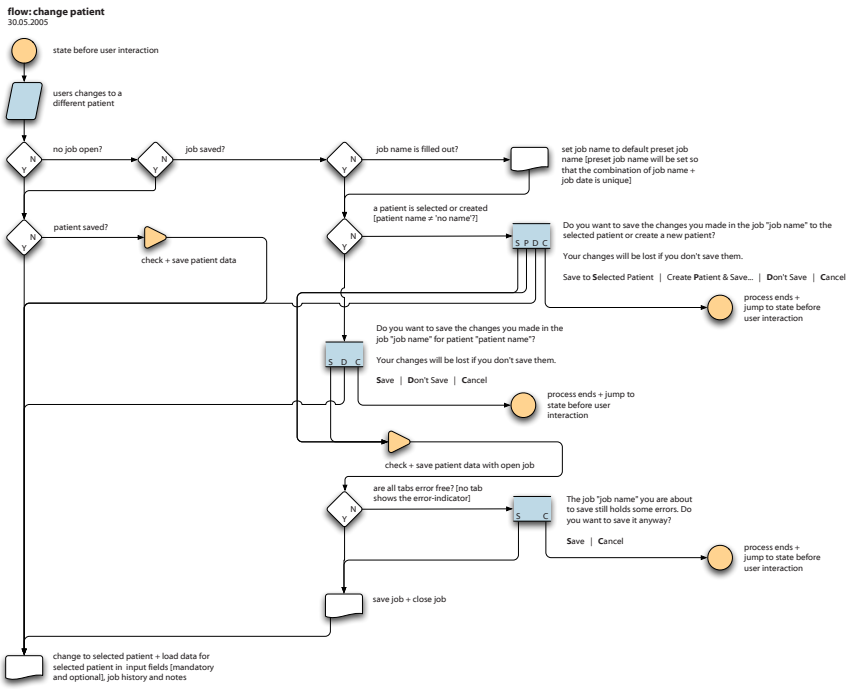


Figure 32: Flow for selecting a different patient.

Flows in combination with screenshots in the project dpm

Normally we use flows to specify the design details for the engineers. Not so on dpm, here we used flows to communicate also with product management, product designers and engineers after the exploration phase.

It took a while to find an appropriate syntax to build the flows. We came up with a combination of an abstract language together with a representation of the buttons positioned in such a way that they almost presented the actual layout on the device. With this approach we were able to kind of simulate the product. By drawing flow by flow for the different states, we not only figured out loose ends in our concept, but got a feeling of the many constraints – such as a small screen and very few buttons – we had to face. In parallel to the flows we developed the screen design. By doing this concurrently we incorporated issues from both sides. We included the key screens very early in the flows (figure 33). The remaining ones were added once they were almost stable. Otherwise the effort to keep them up-to-date would have been to big.

Including screens was very important for discussing features and issues with product management. We arranged all flows for the different states of the dpm on one large sheet and printed it out. The printout for the presentation and discussion was about 6 m long and 60 cm high (figure 34). All team members gathered around a long table. With this setup we were able to both focus on detailed flows but also to quickly jump between different states without losing the overview.

play state | philips

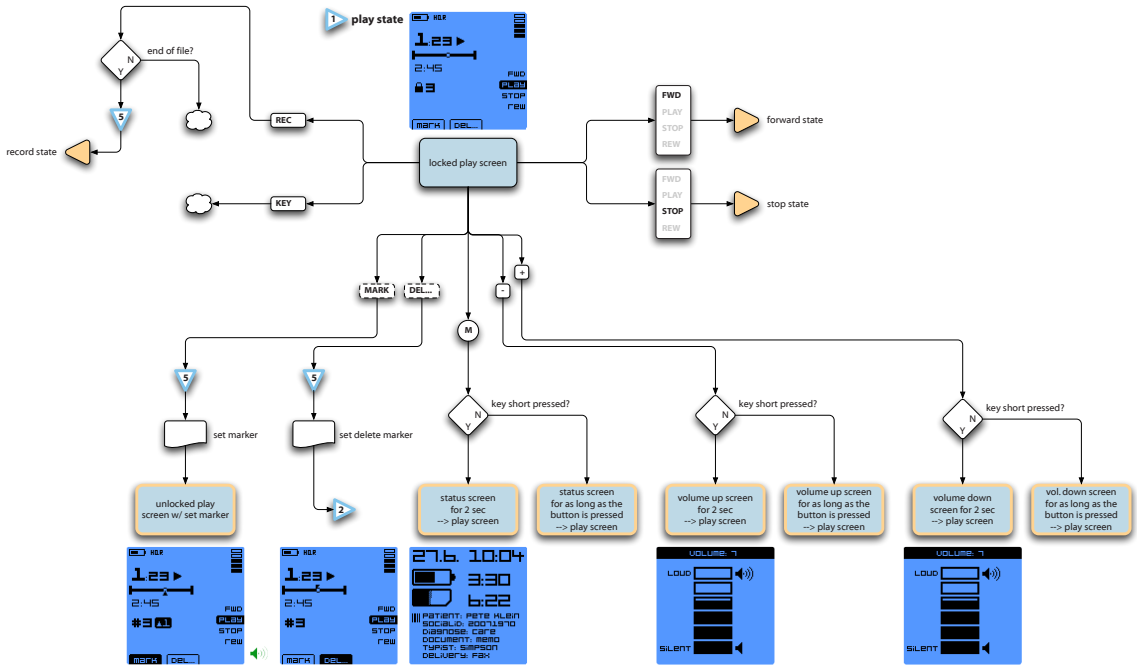


Figure 33: Detailed flow for the play state of the dpm9600.

Flows, screenshots and a physical mockup of the dpm proved to be good communication tools. The flows additionally served as a basis for the engineers to implement the functionality.

Pixel accurate screens

Probably the most common artifacts for interaction designers in a project are screen designs of the key screens. Often those are even the only artifact which is used to transfer the design to the engineers. As important as they are, they only represent the static information of the design. The interactive or dynamic facets can at best only be transported rudimentary. For those the other artifacts are needed.

Crucial to the screen designs is that they are pixel accurate when handed over to the engineers for implementation. They *remove all ambiguity* (Walters 2008), as Michael Lopp put it at a presentation at the last *South by Southwest conference 2008*, where he shared a little peek on Apple's process for developing products. Although it takes a huge effort building them it is worth every single pixel, because they avoid the correction of mistakes later on (Walters 2008). Furthermore they avoid also conflicts between team members. For an engineer it is hard to distinguish if some misalignment is on purpose or by mistake. We had such a situation in the Audi website-project. An engineer *corrected* a misalignment in the screen design and got blamed by the designer for doing it. Of course, he then sticked to the designs. But he did not reckon with the next designer, who

Figure 34: Discussing the behavior of the dpm9600 in front of an overview of all states and flows, which provides easy access to all details. The printout is 6 m long and 60 cm high.

blamed him for *not correcting* an *obvious* mistake, and that he should not be so pedantic. Which shows that it is the responsibility of the designer that the screens are accurate.

Additionally to the plain screen designs we provide a version with dimensions. On early projects we put dimensions on each element, but this took a lot of effort. On recent projects we just overlay the screen design with a grid with rulers on each side (see figure 35). The basis for the grid is defined in the user interface style guide.

The engineers liked this approach even better, because almost all development environments support grids, and then they only need to enter the coordinates which they directly take out of the design. Using the grid instead of the detailed measurements saved us a tremendous amount of work, and also helped us to double check the designs.

Dimensioning only works for fixed sized applications. For scaleable windows we still use the old way with measurements relative to certain anchor points.

Using actual content and naming is another important factor for the screen designs. Moreover the screen designs have to be tested with the longest and shortest copy for all elements used. Who has not been in a project in which the problem with the long button-labels was discovered at the very end? For the navigation-tabs in figure 35, this was solved by a design rule: as soon as not all navigation-tabs fit on the screen anymore the widest navigation-tab has to switch from single-spaced to double-spaced labeling. This rule is captured in the user interface style guide.

Should designers directly implement the interface within the implementation environment?

Hansson (2008) and Fried (2008) argue that the designers should skip pixel accurate designs and directly do the design within the implementa-

tion environment. They argue that the designers should work with the material the product is build of (in their case for web-applications in HTML/CSS) and not on an abstraction of it. Only by dealing directly with the constraints of the material (code) *makes a design feel native* (Hansson 2008). Moreover, doing screen designs is *repeating yourself* (Fried 2008). The designer builds the user interface in a graphical tool, and the engineer has to transform it to the implementation environment.

In the course of negotiating a new project our clients typically ask whether we are also implementing the user interface. We rarely do. The banal reason for this is that we are not proficient in every implementation environment. On some of our projects it is even not clear with which technology the product will be implemented.

In general I believe, or better hope, that in the future better and more flexible interface building tools will replace pixel accurate screen designs as well as parts of the user interface style guide, such as the detailed behavior of custom controls.

On the other hand I see some problems with this direction: Designers would stick just to the standard controls of an environment, because of the lack of skills or even because out of fear, that it would take some effort on their side to implement such custom controls.

Another issue regards reducing interaction design or the design of the product in general to screen design.

Furthermore this cuts back interaction design (or the design of the product in general) to mere user interface design: Over the last years software engineering has evolved, at least in some cases, from seeing design as just creating a usable interface between human and machine to the understanding of an overall user experience, of which the actual user interface is just one part. We have been in a project using *Microsoft Expression Blend* and XAML and there the designers' job got reduced to only creating the screen design. Design was nothing more than a beautification job. Of course, this is not only the case when such tools are used, sadly this is the reality in many companies. It is a mind-set issue about the general understanding of design. But tools like those lead managers and engineers to believe that design is covered with such an integrated approach.

Microsoft Expression Blend
is a design tool to create rich user interfaces for both desktop applications and the web. It is a WYSIWYG editor for designing XAML-based interfaces. (Microsoft 2008a)

XAML
Extensible Application Markup Language (XAML) is a XML-based language for declarative application programming created by Microsoft. It separates the code for the user interface from the application code. (Microsoft 2008b)

I do appreciate tools which support an integrated approach of transferring the design into the actual interface such as user interface builders or Microsoft Expression Blend. But it is important to establish a mutual understanding within the team, on how those tools are seen in the overall process. In my perspective the tools mentioned above are only a tool to create one artifact among many others.

The question remains: do designers have to have the skills to implement part of the design – the user interface design – by themselves? In

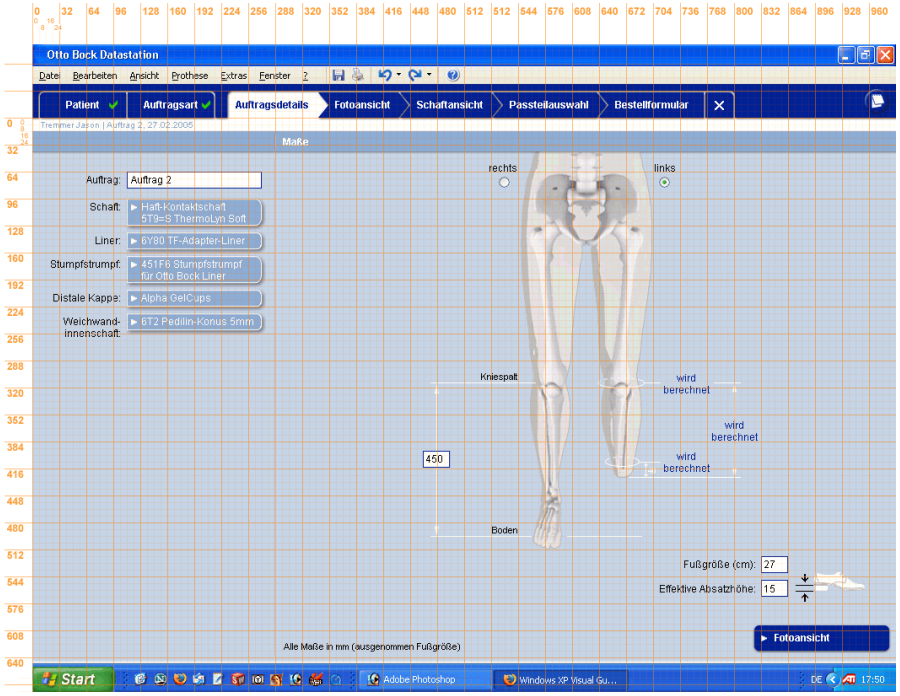


Figure 35: Pixel accurate screen with dimensions for the project fuse.

my opinion interaction designers need to have a profound understanding of the material they are working with, but they do not need to be able to implement the interface in production quality code. I much rather prefer designers to spend their resources for exploring, sketching and designing the overall product than on polishing the code for the user interface.

Interaction design style guide

The interaction design style guide, also known as user interface style guide, specifies guiding principles for the various interaction elements used, their usage and arrangement, as well as the general look & feel of the whole application (see figures 36 and 37). It is based on the corporate identity style guide of the company and the general interface guidelines the application is running on. The user interface style guide is:

- A tool to ensure consistency of de-facto standards, guidelines and conventions of the environment the product is living in terms of, terminology, interaction behavior, look & feel, error messages, etc.
- A repository for design guidelines and standards.
- A training material for new team members, designers and engineers.

The user interface style guide is usually one of the artifacts requested by the client and/or engineers. Often a project starts with the request for a style guide. But doing just the style guide without understanding the users or the context of the application, or without involvement in the interaction design, would just be a paint job. Mostly we are able to convince the

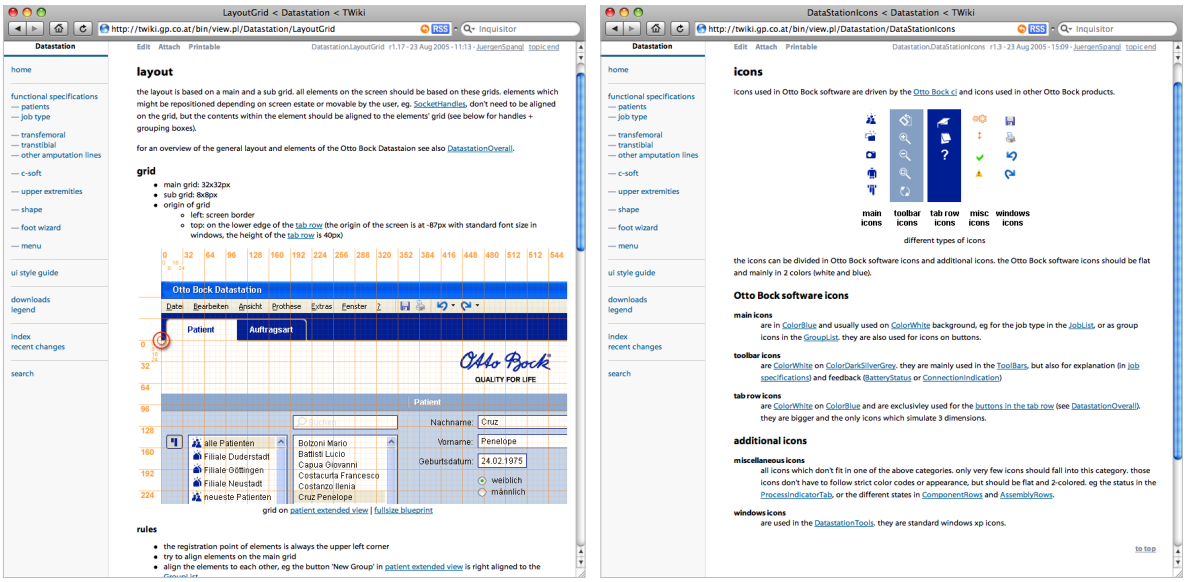


Figure 36: Interaction design style guide: General specifications for the grid and icons.

client of the need of those prerequisites. But even if we fail, it is better to be involved in the creation of the style guide, than not at all.

As discussed earlier, we used an internal wiki to document our decisions during the project fuse. During the shape-phase both the visual language in alignment with the behavior of the applications got more and more specific. In the last versions of the prototypes not only the behavior was tested, but also icons, font sizes and colors. Those findings were collected briefly in the style guide. They then got detailed in the complete-phase in the user interface style guide for the client.

Interaction design pattern library

Interaction design pattern libraries (also known as user interface pattern libraries) are based on the idea of design patterns introduced by Christopher Alexander in the field of architecture (Alexander et al. 1978). He defines a pattern as follows:

> *Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution. […] Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over.* — Alexander et al. 1978

The concept of design patterns were first mentioned in the field of human computer interaction by Norman & Draper (1986), but they only really took off with the workshop *Putting it All Together: Pattern Languages for Interaction Design* at the CHI'97 (Erickson & Thomas 1997). Welie & van der Veer point out that the use of patterns is slowly gaining momentum

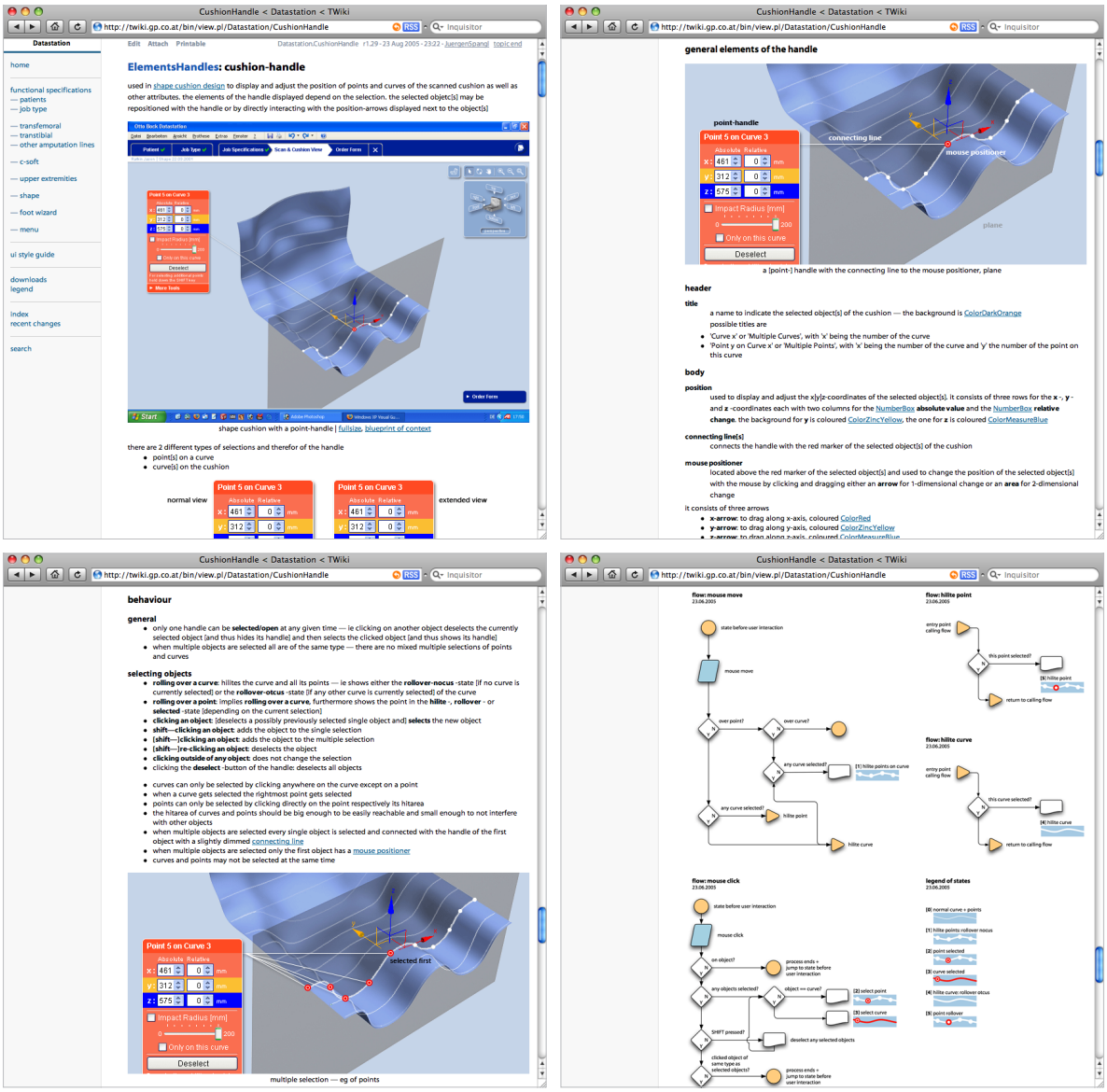


Figure 37: Interaction design style guide: Detailed description of the behavior of a custom control, the cushion-handle, for manipulating 3D-data.

in the practice of interaction design and also its related fields, such as web design (Welie & van der Veer 2003). Many collections of interaction design patterns are published in books (e.g. Borchers 2001, Graham 2003, Tidwell 2005a, van Duyne et al. 2006) or online (e.g. Laakso 2003, Tidwell 2005b, Snow et al. 2006, Welie 2007, Toxboe 2008, Yahoo! 2008) (figure 38).

Welie states that interaction design pattern libraries are similar to guidelines in the interaction design style guide, but yet different (Welie 2001):

> *The purpose of guidelines is to capture design knowledge into small rules, which can then be used when constructing new user interfaces. A pattern is supposed to capture proven design*

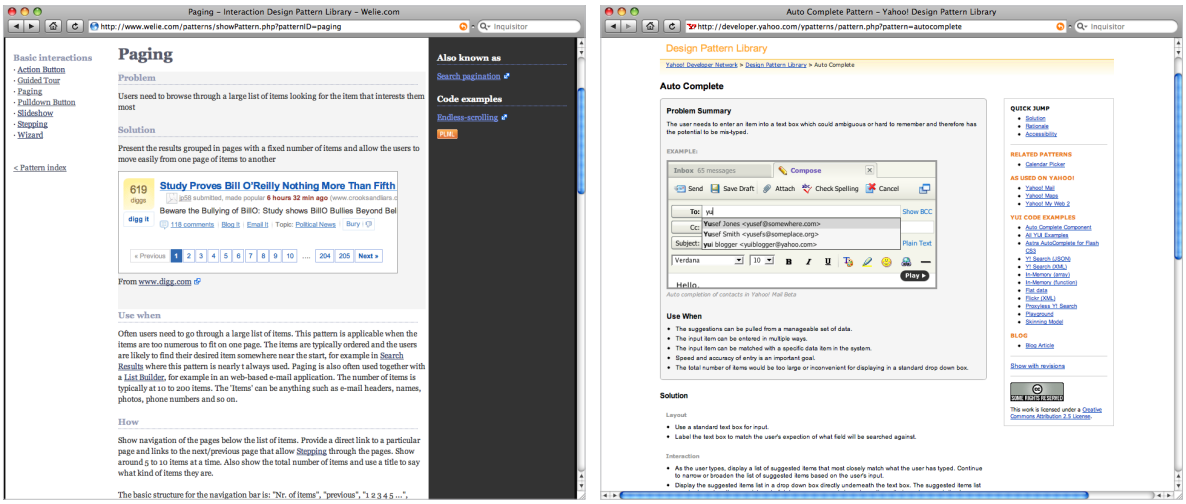


Figure 38: Two examples for a pattern description: Paging pattern (Welie 2007) and Auto Complete pattern (Yahoo! 2008).

> *knowledge and is described in terms of a problem, context and solution. […] patterns represent pieces of good design that are discovered/uncovered empirically through a collective formulation and validation process, whereas guidelines usually are defined normatively by a closed group.* — Welie 2001, p 94

Most of the above mentioned pattern libraries use a similar template for describing the patterns:

- Pattern name
- Problem statement
- Solution
- Use when (the context)
- Why (the rationale)
- Examples

Sometimes, especially for web design patterns, also the code for the implementation of the pattern is provided. In this case the designers and engineers work hand in hand in building up the library.

In our projects we mostly use a mixture of both interaction design style guide and interaction design pattern library. Although we do not stick to any formal description, such as the three minimal parts of a pattern (a problem, its context and a solution to it), we follow it roughly. On the other hand our interaction design style guides are more than just guidelines, because we provide examples and a rationale to them. Dix et al. (2003) report that guidelines have problems when used in practice, because they are too abstract and can be difficult to select and interpret. We generally use the term interaction design style guide for our mix of formal guidelines, user-centered design principles and the detailed description of custom controls and their usage (this could probably be

called the description of a pattern), because most of our clients are used to this term from the corporate identity style guide.

Specifications as boundary object

Engineers are used to the customer requirement specifications in plan-driven environments. The specifications as described in the previous chapters are extended customer requirement specifications, driven by the user needs and cover the application in detail.

The counterpart in agile environments are user stories. With the involvement of interaction designers the user stories describe the needs of the user in more detail, and not only from a customer perspective, which is often the case.

Specifications as a tool for reflection

The specifications are not only a (the final) deliverable of the designers to explain the product (or functions in agile environments) in detail to the engineers. The process of writing and drawing the specifications are, in my opinion, a tool for the designer and team to reflect on the designs. For example, while drawing the flows for a certain detail of the product, the designer discovers conditions which were not taken care of at an earlier stage. Not taking care of details in an early stage is done on purpose and a major differentiator to engineering. Focusing to early on all possible exceptions and on every detail would block tinkering and exploration (37signals 2006, p 40). At the beginning design is fuzzy and imprecise. Sketching would be impossible if one tries to solve every detail. This imprecise approach at the beginning is a key characteristic of design. Later on it is about reduction, making decisions (Buxton 2007, p 145). Now the design process is about the many details, which only get discovered and defined at this stage. Sometimes, although rarely, this might demand a redesign of a complete feature or even finding a new approach.

The problem is that documenting is not a very popular task among designers. It is often seen as just writing down what is anyway already expressed in some screenshot. Designers do not see the reflective quality of documenting. Good products rarely just live from fancy new approaches or metaphors alone – it is the details that matter in the day to day routine.

A few years ago we did the interaction design for a financial product. The goal of the application was to support farmers and young entrepreneurs with their accounting without the need of being an expert in taxes and business. We came up with a new approach to accounting in which we combined the view for non-experts with the one of experts. The client liked it. It performed well on usability tests during the design process. Although the application got positive reviews in general, I do not see it as a good product. It is good enough to get good reviews, because in those

conditions it is not usually used as it would be used in a daily routine. It was even in such a state that I did not use it as accounting tool for our company.

What did not work? – Many details. The overall concept worked great. But the details did not work. Details such as entering the date. The user had to stick to a given format and every detail of the date had to be entered. There was not such a thing as just entering a number and it gets translated into the day of the current month. Or simply using the '+'- and '-'-key in the date-field to increase or decrease the date by one day did not work. Another issue were presets for accounting records. They were always set to some default instead of using the information from previous records for the same payee. Those were all functionality standard in the products of some of our competitors. I was used to this behavior from the software I previously used. Although the old software did not have some of the nice features of our software, I sticked with the old and known because of the smart and simply way it allowed me to enter my data.

What had happened? – The project had only a small budget for interaction design, therefore we delivered the design only by the way of screenshots and very limited specifications of the features. Most of it was communicated in presentations. I believe that, if we would have done the specifications in more detail we would have stumbled over those issues. No one thought about presets or detail operations. Not doing the specifications was not only because of budget reasons. We were too engaged in transporting the overall metaphor and approach and thus forgot about the details, or maybe we thought they were common behavior within financial applications anyway and therefore did not need to specify it.

We learned our lesson, and figured out that caring about details is an integral part of designing a product. Depending on the project (in terms of development environment, budget, size, team, etc.) we define the level and type of the specifications. For the final product, to hit the user, someone has to decide everything, and I (as an interaction designer) much rather prefer being involved and taking responsibility for it, than to just leave it to the engineers. But it is important that for those decisions the whole team is involved.

11.5 Technical documentation

Technical specifications, such as technical requirements specifications, software architecture and entity-relationship-diagrams (ER-diagrams) are already available in most of our projects. Even in projects with design aware companies we are mostly brought into, only after the first technical concepts have already been developed. Additionally often documentation from the previous version of the product exists.

For designers those documents are very helpful to get a better understanding of the way the engineers tackle the problem. Moreover they provide a basis to better understand the constraints, which is very helpful when it comes to arguing whether something is possible from a technical point of view. Needless to say that this requires designers to have a rough understanding of how software is built (cf. chapter 10.3, p 52).

Interestingly when we request such documents the engineers are often very reluctant to hand over those documents. I can only assume what the reasons might be, but in my experience the formal documentation is often missing and only present in the engineers knowledge. On fuse we got a rough requirements document, but we were not able to manage to get an ER-diagram for the database for the component selection. We were turned down with the argument that we would not understand it, because it was only a rough document. But only with the requests and usage and understanding of technical terms we were able to gain credibility.

In my opinion it is important to exchange artifacts in both directions rather than only providing interaction design artifacts to the engineers.

Part 4

The Catalyst Kit

Part 4 introduces the Catalyst Kit – a toolkit to encourage design thinking by focusing on the collaboration within the team and by providing methods and tools for improving the transition from the first concepts to the final product.

When I started my thesis my focus was to investigate the collaboration between designers and engineers to discover methods and practices for delivering a product which creates a good user experience. This still is the core of this work. Being used to developing products I wanted to find a way to transport the findings of this thesis in a more tangible way, rather than hidden within 100+ pages.

In this chapter I will describe the Catalyst Kit. It is a toolkit built on the findings of the previous part, a way to enhance the collaboration between different communities of practice, namely designers, engineers and business people in a project.

12 Why the name *Catalyst Kit*

In an interview on his newly designed lamp *Solar Tree* at the *Museum für angewandte Kunst* in Vienna, Ross Lovegrove (2007) answered the question *What it means to him to be a designer* with:

> *I feel not as a designer, but as a catalyst between nature and objects.* — Lovegrove 2007

Inspired by his use of the word catalyst in the context of design I argue that the pool of methods and tools found are *catalysts* in projects between the different team members. Also the non chemical definition of catalyst

in *Merriam-Webster's Online Dictionary* (2008), *an agent that provokes or speeds significant change or action*, backs the usage of the term. Hence my proposed toolkit will be called *Catalyst Kit*.

13 Inspiration for the catalyst kit

The main influence for the catalyst kit are the *IDEO method cards* (IDEO 2003). Before giving a brief explanation of the IDEO method cards and my experience with using them, I would like to bring additional sources of inspiration for the catalyst kit.

Working with tangible artifacts – such as cards, post-its, photographs and screen shots – and arranging them on a wall for reflection in a collaborative setting is typical for design (Blevis et al. 2005) – but also used in other disciplines, such as project management and engineering. For example in agile environments user story cards (Beck & Andres 2004) and estimation cards (Cohn 2005) are a common tool to encourage discussion and quickly get a shared understanding. In plan-driven environments CRC cards are used for object oriented development (Wilkinson 1995). In relation to the interdisciplinary view on projects (p 68) Scott Berkun states that making facts accessible and visible is almost as important as the fact itself:

> *Almost as important as its strategic planning value, using a Venn Diagram [see figure 13, p 69] can defuse perspective bias of engineers or marketers. It helps teams see overlapping points of view, rather than only competing ones. Early and often during project-planning discussions, this diagram or something like it (e.g. a diagram that includes a list of potential goals from each perspective) can be used to frame suggestions made by people who have bias toward one view of the project.*
> — Berkun 2005, p 50

Additionally our experience with personas on fuse (p 76) was an inspiration. Providing an artifact not only as a (digital) document, but in a tangible form (in this case laminated cards of the personas), made it visible and easily accessible throughout the project. The simple lamination alone made it a concrete, tangible artifact and thereby easier to refer to it. The team members could take their set of personas and hold it up to use it as a reference for everyone in discussions.

Figure 39: IDEO method cards.

IDEO method cards

The IDEO method cards (IDEO 2003) are a set of 51 cards with different methods for user research. They were collected and created with the idea to make numerous techniques accessible to a larger audience. The cards are grouped into four categories representing approaches to empathize with people (IDEO 2003):

- ***Learn:*** *Analyze the information you've collected to identify patterns and insights.*
- ***Look:*** *Observe people to discover what they do rather than what they say they do.*
- ***Ask:*** *Enlist people's participation to elicit information relevant to your project.*
- ***Try:*** *Create simulations to help empathize with people and to evaluate proposed designs.*

Each of the cards holds a short explanation of the method on one side: how and when it can be applied and short example of how it could be used on a real project. On the other side it is visualized with an illustration or image.

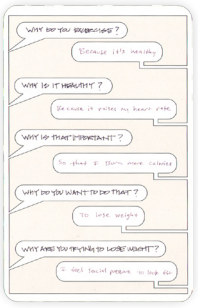
Here is an example of one card for each of the four categories (IDEO 2003):

Learn: Error Analysis

- ***How:*** *List all the things that can go wrong when using a product and determine the various possible causes.*
- ***Why:*** *This is a good way to understand how design features mitigate or contribute to inevitable human errors and other failures.*
- ***Example:*** *The IDEO team used error analysis on a remote-control concept in order to maximize the functionality of each button's size, shape, and texture.*

Look: Personal Inventory

- ***How:*** *Document the things that people identify as important to them as a way of cataloging evidence of their lifestyles.*
- ***Why:*** *This method is useful for revealing people's activities, perceptions, and values as well as patterns among them.*
- ***Example:*** *For a project to design a handheld electronic device, the IDEO team asked people to show and describe the personal objects they handle and encounter every day.*



Ask: Five Whys?

- ***How:*** *Ask »Why?« questions in response to five consecutive answers.*
- ***Why:*** *This exercise forces people to examine and express the underlying reasons for their behavior and attitudes.*
- ***Example:*** *»Five Whys« was used when interviewing dieting women around the US to understand their attitudes and behaviors around weight loss.*

Try: Predict Next Year's Headlines

- ***How:*** *Invite clients to project their company into the future, identifying how they want to develop and sustain customer relationships.*
- ***Why:*** *Based on customer-focused research, these predictions can help clients to define which design issues to pursue in product development.*
- ***Example:*** *Designing an intranet site for information technologists, the IDEO team prompted the client to define and clarify their business targets for immediate and future launches.*

I use the cards, among other situations, for preparing client workshops and also in the workshops itself. I take the whole set and flip through the cards, single out interesting and useful cards, ones which could be helpful to encourage the workshop participants sharing their views or help them to put themselves in the position of their customers. The selected cards are then used in the workshops to introduce the methods and discuss it.

Although I am perfectly familiar with most of the represented techniques, the cards help me to recall them easily and they act as an inspiring tool by bringing them back into my mind. Furthermore they are an excellent way as a starting point in introducing methods to people who are not familiar with user research.

14 The catalysts

The goal of the catalyst kit is to make different viewpoints visible and encourage discussion to gain a mutual understanding and to shape the process or approach most appropriate for the current project.

In the process of designing the catalyst cards many ideas arose. I just like to present two of them:

- Using one side of the card for the designers and the other side for the engineers. The disadvantage of this approach is that it brings the separation back in, and would not transport a holistic approach. Additionally it includes only those two disciplines, but excludes all other disciplines involved in a project. Furthermore the versus-approach is only viable for the artifacts.
- Foldable cards, similar to small tube or city plans. The idea was to have a short explanation on one side, and an image or graphic on the other. When opened it holds a more elaborated explanation and more examples. The advantage of having all information nicely together comes also with some problems: using it in a stack and flipping through the cards becomes difficult, because of the fragile and bulky structure of the folded card. But flipping through the cards and exchanging it within a group setting is a major element of the catalyst kit. In addition, the foldable cards are not as sturdy as simple cards and this might then become a reason for not using them often.

In the end I came back to a representation in plain simple and sturdy cards similar to the IDEO method cards.

The catalyst kits consist of two entities:

- **Proposed catalysts:** Those are the tools and methods found in the previous part *Teamwork: designers & engineers* (p 31). Each catalyst proposed consists of a short explanation of the tool or method and states in which context it is used best. Furthermore a reference to a more elaborated explanation is given.
- **Empty catalysts:** A stack of empty cards is provided to encourage the creation of new catalysts appropriate to the current project.

Those catalysts could then be transformed into printed catalysts and in this way become proposed catalysts for the next projects.

Furthermore each catalyst card belongs to one of the following categories:

- Process
- Collaboration
- Artefacts

Following is a description of all catalysts. The actual catalyst cards can be found in the *Appendix* (p 155).

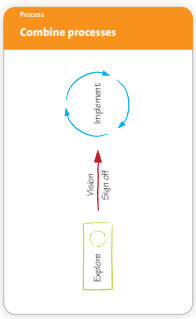
14.1 Question the process

How
Question the process: Does the currently followed process still make sense? Are all viewpoints included? Does the process support a holistic approach? Are business, design, and engineering equally represented? Do all team members in the project fully understand and comply with the process?

Why
A good process should be open for change. It should provide mechanisms to constantly review itself. Designing the process is also a part of design. Team members need to feel responsible for the results and the impacts processes have on projects, therefore they need to fully understand and support the process.

Category: Process
Further information: Summary – Process (p 45)

14.2 Combine processes



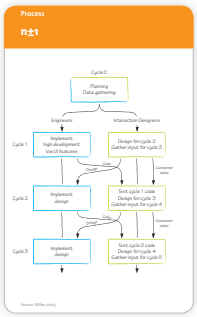
How
Combine elements from plan-driven and agile environments. Start with a user-centered phase with user research and explore different metaphors and concepts up-front before starting implementation. This could even include building some prototypes iteratively to get a better understanding of the context and feeling for the product. Once agreement is established for an overall concept (vision) it builds the basis for the user stories. The implementation phase then uses an agile approach. Remaining design

issues are clarified and detailed during the iterations. Additionally, the quality of the design is reviewed and evaluated with usability tests.

Why
Working out the concepts up-front provides the whole team a perspective (vision) for the product. Furthermore design is not limited by the technical implementation, and hereby faster and can factor in user feedback earlier. But compared to plan-driven environments the combination gives more flexibility in responding to changing requirements and to findings from the usability tests. Time and budget is saved, because design detailing is only done for those features, which are actually implemented – and with the insight of how the actual software is working.

Category: Process
Further information: Up-front design (p 36), Our experience with processes (p 41)

14.3 n±1

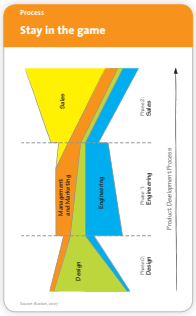


How
How to better include design in agile environments? Designers should support the product owner (customer) by describing and choosing the user stories for the next iteration, based on the findings from user research. The design for the features described in the user stories should be finished before the iteration for implementation starts, meaning it should be one iteration ahead (n-1). The user stories get implemented in iteration n. The following iteration (n+1) is used for usability testing and refinement of the features, just implemented. Hence the designers will – in every iteration (except the first and last one) – design and detail the user stories planned for the next iteration, and conduct usability tests and refine the user stories implemented in the last iteration.

Why
n±1 involves not only the customer, but also the user. It filters and interprets the user needs into features and user stories and furthermore saves time by combining the user research and usability test into one session with the user. Engineers can focus on the technical aspects, without loosing time by neither doing the design themselves nor by waiting for the designers who are working in the same iteration on the same user story.

Category: Process
Further information: Integrated iterative interaction design (p 38)

14.4 Stay in the game



How

All parties – business people, designers, and engineers – should be involved during the whole product development life cycle. Depending on the phase of the project, the involvement of each of the parties might differ – at the beginning it usually leans more towards design, later on engineering takes over.

Why

Engineers are needed from the very beginning on to provide technical constraints and to check the technical feasibility of the design concepts. During later phases the involvement of the designers is important to assure that the implementation conforms to the concepts and provide designs of upcoming issues.

Category: Process
Further information: Design in plan-driven environments (p 33), Our experience with processes (p 41)

14.5 Pair design/programming

How

Pair design/programming is a reference to pair programming in extreme programming. Instead of two programmers pairing up, a designer and a programmer sit together in front of the computer and tinker with certain aspects of the user interface and behavior of the application. This is usually done on a branch of the actual product and without adhering to programming guidelines. It is a form of prototyping done on a branch of the product.

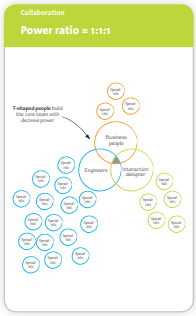
Probably most useful in agile environments, but it can also be applied within plan-driven environments during the implementation phase to decide arising detailed issues.

Why

Working with the actual material is often needed to decide on certain design details. Building prototypes for every detail is too time consuming and expensive, and often design issues not arise until implementation.

Category: Process
Further information: Our experience with prototypes (p 95)

14.6 Power ratio = 1:1:1



How

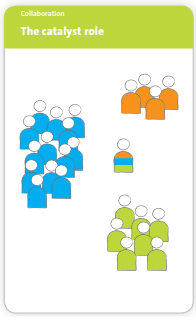
Balance the power within the team 1:1:1 among the involved perspectives: business to technology to user. The power is democratic within the three perspectives and not the whole team. The balance can be established by a core team, consisting of a member of each perspective.

Why

The interdisciplinary view of the project team is crucial for developing a product which delivers a good user experience. Every perspective counts equally. Over the course of the project the power might shift between the perspectives, e.g. at the beginning the power of the user perspective is higher, during implementation this may change towards technology. But still all parties are involved in the decisions. Over the lifespan of the whole project the power ratio is balanced.

Category: Collaboration
Further information: Balanced teams (p 68)

14.7 The catalyst role



How

Facilitating consensus among the different viewpoints is the main task of the catalyst role. The individuals who take this role have to be able to understand the different languages of all perspectives involved. People of the core team need to have the capabilities of the catalyst role. On larger teams the catalyst role could even be dedicated to a single person and be this persons single role.

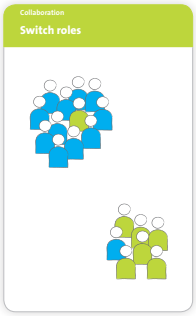
Often the designers end up or take on this catalyst role, because their skills in observing, listening, synthesis, and in creating artifacts that people can relate to are also relevant to facilitate consensus.

Why

Delivering a good user experience is the responsibility of everyone in a project team and calls for the input of all disciplines involved. Synthesizing the divergent languages and often contrasting viewpoints of all disciplines is key to satisfy this demand.

Category: Collaboration
Further information: The catalyst role (p 67), Capabilities of an interaction designer (p 52)

14.8 Switch roles



How

Switching roles over a short time frame together with working along your team members from a different discipline helps exchanging knowledge and learning each others' methods and tools. This is an excellent tool to gain the capabilities needed for the catalyst role.

Why

Understanding each others' languages, methods and techniques is the basis for finding consensus rather than compromise. Only consensus will deliver a product with good overall user experience.

Category: Collaboration
Further information: The catalyst role (p 67)

14.9 The user experience is your responsibility

How

Take responsibility for the results and impacts processes have, for your decisions and artifacts in a project. For designers this means taking responsibility for their concepts and staying in the project until they reach the user. For engineers this means taking responsibility to get involved early in the project to inform the design team of constraints and support them by checking the technical feasibility early on. Furthermore it is their responsibility to implement the product as close as possible to the concepts.

Acting in a designerly way is the responsibility of everyone, but bringing design thinking to the team is often the responsibility of the designers. Take the responsibility, for a better user experience.

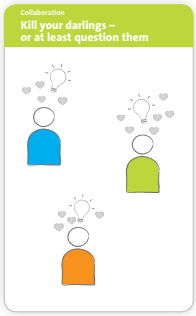
Why

Who owns user experience? – The team.

A good user experience can only be delivered if everyone within the team contributes his share. Hence everyone should accept and carry the responsibility for the user experience of the product.

Category: Collaboration
Further information: Who owns user experience? (p 49)

14.10 Kill your darlings – or at least question them



How

Do not fall in love with your ideas. Encourage others to question them. Try to get to the bottom of them.

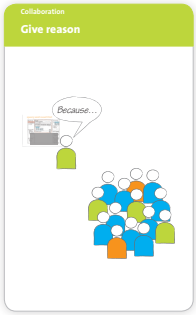
On the other hand: fight for your concepts. Do not throw them over board too quickly, because you get the feedback that they are technically not feasible – question also such a feedback. But be honest with yourself: are you fighting for your idea because it is your idea – or because it is in the interest of the user?

Why

Getting emotionally too attached to your concepts, especially your first ones, can block other ideas and thus can put the project at risk. On the contrary, just following force of habit and using the same solutions over and over without questioning them will never allow for innovation, no more than using concepts only because the development environment provides them.

Category: Collaboration
Further information: Generation three: doing for the sake of knowing (p 14)

14.11 Give reason



How

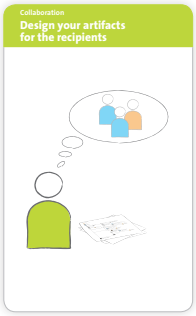
Communicate your judgements and your design rationale to your recipients. Describe your drivers, motivations and reasons during the design process. Also explain how you are doing it. Let others take part in your methods and approaches.

Why

Giving reason helps the team to build up a common ground in understanding each others' needs to be able to provide valuable feedback and input to the ideas and concepts. Furthermore it transports the approach of design thinking.

Category: Collaboration
Further information: Getting your point across (p 55)

14.12 Design your artifacts for the recipients



How

The artifacts should always transport their intention: be it for gathering feedback (e.g. prototypes) or to communicate decisions (e.g. functional specifications).

Prepare the deliverable for the recipients as you would design the product for the users. This also transports the quality of the design. If designers produce high quality deliverables, also the engineers and the rest of the team will create high quality deliverables, and this will result in a high quality product.

Take responsibility for your artifacts, especially for the specifications. Someone has to decide on the details – if the designers are not doing it then the engineers have to do it, otherwise they cannot implement it.

Why

Selling the concept to the client is an important part of design. But this is also true for selling it to other team members. Selling does not only cover the presentation, but must be extended to the way your artifacts are edited.

Category: Collaboration
Further information: Getting your point across (p 55)

14.13 Specifications – clarify them

How

Specifications summarize the decisions made during the design process and define the product in different ways (see the artifact cards) in an implementation friendly style.

As such it is important to gain an understanding of the engineers on which parts need to be captured in which level of detail. In this regard it is useful to compile a small part of the specifications and then discuss them with the engineers – test your specifications with the engineers (the users in this case) as you would test your designs with the user.

In agile environments the specifications are usually on a per feature basis and not as detailed as in plan-driven environments. They are elaborated user stories including detailed screens and flows. But again, the level of detail has to be agreed upon within the team.

Why

It is in the interest of the team – but in the responsibility of the designers – to shape the transformation of the design into implementation as smoothly as possible.

Category: Collaboration
Further information: Specifications (p 103)

14.14 Personas

How

Make your users part of the team by using personas. Bring them to every meeting and share them among the team members.


Artifacts
Personas

Personas:

- are a portrait of a typical user ideally based on user research data as well as input from the client.
- are hypothetical.
- are archetypical and not stereotypical.
- represent important demographic data.
- live in a social context.
- have characteristics and goals.
- are *alive*.

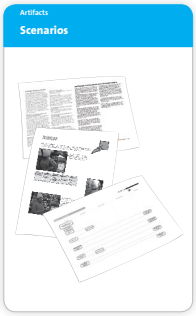
Why

Personas are a tool to give the anonymous users a face and to encourage team members to distinguish and also articulate their own ideas apart from the user's view.

In plan-driven environments personas can stand in as actors for use case models. In agile environments personas take the user role within the user stories.

Category: Artifacts
Further information: Personas (p 73)

14.15 Scenarios



How

Scenarios are descriptions of people and their tasks, they are a mixture of the currently applied procedures and the wishes expressed by the users of how a future system should or could work. They typically focus on happy day scenarios and do not deal with all exceptions. Scenarios are presented in many different ways, such as told narratives, textual stories, storyboards or short videos.

Designers use scenarios to discover new grounds and to transfer the knowledge gained during user research to the rest of the team.

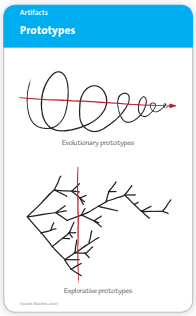
Why

Scenarios and use cases: use cases can be developed from scenarios. Compared to scenarios, use cases describe all possible paths, and are written in a more formal style. Use case models give an excellent overview of the whole system, but do not transport the priorities of the different cases – this is covered by scenarios.

Scenarios and user stories: user stories are usually much shorter than scenarios and focus on one detailed function. Scenarios provide product managers and engineers with a real world context and a bigger picture of the product.

Category: Artifacts
Further information: Scenarios (p 80)

14.16 Prototypes



How

Design usually uses throw away prototypes for explorative purposes. Prototypes act as both: a tool for usability testing and an inquiring material to explore possible interactions. Depending on the current project phase and need, the most useful type of prototype should be chosen, ranging from paper prototypes to highly functional ones, which mimic the intended product behavior in great detail.

For engineering evolutionary prototypes are more common. Mostly built within the very development environment also the final product is built with. They often act as proof of concept.

Discuss the intention of the prototype and exchange the findings of its application to establish a mutual understanding of the different types of prototypes used in the project.

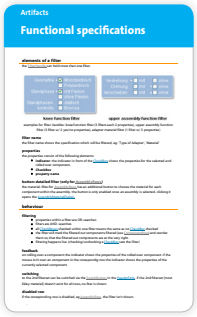
Why

Prototypes are a way of communicating with ourselves, with the users, with the client and with the team members to gather important feedback and findings. Prototypes also often serve as a living guidance for the engineers while implementing the product.

Category: Artifacts
Further information: Prototypes (p 87)

14.17 Functional specifications

How



Functional specifications describe the functionality and interaction concepts and are based on the interaction design style guide. Elements are specified in detail in their respective context of use. They define the interaction and its implications for every element and describe what happens in case of an error.

Furthermore they are often the holder for the other specifications, such as flows and pixel accurate screens.

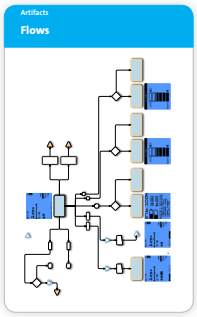
Why

The functional specifications often become the requirement specifications for the engineers. With the distinction that they not only cover the customer requirements, but also the user requirements, flows and pixel accurate screens.

Category: Artifacts
Further information: Functional specifications (p 106)

14.18 Flows

How



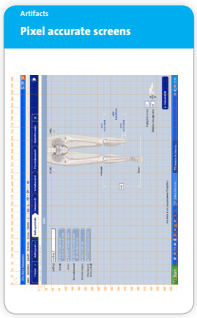
Flows cover the details and exceptions, such as error handling, of a product. The syntax used in the flows is defined to the needs of the project and in agreement with the engineers.

Why

Flows are useful for both designers and engineers. For designers they act as a tool for reflection in later stages, when it is time to decide on the details. For engineers they are a part of the specifications.

Category: Artifacts
Further information: Flows (p 107)

14.19 Pixel accurate screens



How

Pixel accurate screens represent the static information of the design. It is the responsibility of the designer to ensure that the screens are pixel accurate to remove any ambiguity.

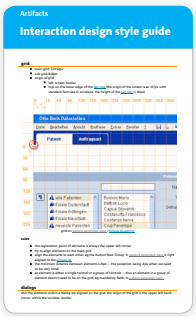
For fixed sized screens/windows a grid overlay with rulers on each side works well. For scalable applications/windows the scalability and arrangement of the elements has to be covered.

Why

Pixel accurate screens remove the ambiguity and are especially important within plan-driven environments. Within agile environments they also should be as accurate as possible, but here the process gives room for a better communication to clarify uncertainty.

Category: Artifacts
Further information: Pixel accurate screens (p 110)

14.20 Interaction design style guide



How

The interaction design style guide, also known as user interface style guide, specifies guiding principles for the interaction elements used, their application and arrangement, as well as the general look & feel. It is based on the corporate identity style guide of the company and the general interface guidelines the application is running on.

A good user interface style guide can only be built on the basis of a thorough understanding of the users' needs and context of the application and the resulting concepts and interaction design.

Why

The user interface style guide provides guidelines for building new components of the product. Furthermore it acts as training material for new team members – designers and engineers – to get them quickly into the metaphors and drivers of the project.

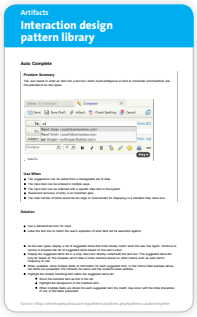
Category: Artifacts
Further information: Interaction design style guide (p 113)

14.21 Interaction design pattern library

How

An interaction design pattern is a best practice for a recurring design problem. The description of a pattern consists of at least three parts: a problem, its context and a solution – but for better understanding should be accompanied by a rationale and at least one example of use. If possible provide a code example for the implementation of the pattern.

A very complete collection of patterns make up a interaction design pattern language or interaction design pattern library.



Why

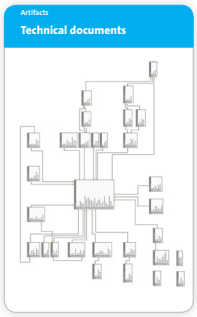
The interaction design pattern library provides an overview on many common design problems and may serve as a learning tool for unexperienced team members – and as an inspiration and reference for all team members.

Category: Artifacts
Further information: Interaction design pattern library (p 114)

14.22 Technical documents

How

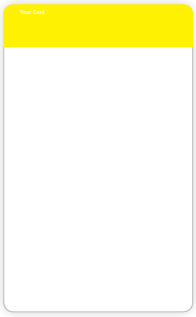
Provide technical documents – such as technical requirements specifications, software architecture and ER-diagrams – also to team members of other disciplines, especially designers. They provide a basis to better understand the technical constraints and to gain insight into how the engineers see the problem.



Why
It is important to exchange artifacts in both directions, from interaction designers to engineers and vice versa, in order to build up a holistic view of the project.

Category: Artifacts
Further information: Technical documentation (p 118)

14.23 Your cards



Well, what to write about empty cards? – Not much, I guess. They are here for *You*. Every time you stumble over a method, technique or tool which worked to encourage collaboration and design thinking in your team on a specific project you take one of the empty cards – your cards – and summarize briefly

- how the method works,
- why you think it is helpful – and in which context,
- and provide examples or references to further information.

On the next project your cards become then a part of the regular deck. When using the catalyst kit, always have some empty cards ready to foster the creation of new ones.

15 Who should introduce the catalyst kit?

In short: *everyone can and should*. Whoever on the team knows about the catalyst kit and thinks it would be a good tool to improve the collaboration within the team.

In my experience we, as interaction designers, were often the ones which questioned the overall process, demanded a more user-centered approach and called for a larger stake in the project. Furthermore the job of the designer is to grasp the users' needs and transfer them into a digital product. The interesting thing is that to accomplishing this designers also need to cater to the engineers' needs. The ideas and concepts have to be prepared in such a way that the engineers are able to implement them. Of course, in an ideal environment this should also apply the other way around and the engineers should be considerate of the designers' needs.

All of the project team are participants and users of the project environment, so everyone influences the experience within this environment. The project team is responsible for their own experience within the project. Because of their knowledge in regards of user experience, designers are a perfect fit to take the lead and introduce new approaches and

to try to do everything to establish a good experience within the project team. I believe that a good experience within the project team leads to a better motivated team – and this leads to better products. And ultimately leads to a better user experience for the user.

In the interaction design community and at our local IxDA face-to-face meetings (Spangl & Haberfellner 2004) many interaction designers complain that they have very little impact on the projects and that they are often the sole interaction designer or a very small group compared to the overwhelming number of engineers in the company. Moreover, interaction design is a young discipline and many of the designers hold the most junior position in project teams compared to the more mature discipline of software engineering. Interaction designers can use the catalyst kit to introduce a designerly approach to the team members. It is a starting point to evangelize design and design thinking.

Often product managers or project managers are aware of the need of designers in a team, but simultaneously lack the knowledge of processes and tools to integrate them in the project. In some of our consulting projects we were asked to not only do the actual interaction design, but to also introduce design processes and design thinking to the project team. Here we made use of the tools and methods summarized in the catalyst kit. Additionally to our role as interaction designer we also fulfilled the role of the catalyst.

To come full circle to the beginning of this thesis, where I mentioned Bill Buxton's call for a *Chief Design Officer* (Buxton 2005) – he would be a perfect candidate to provide the catalyst kit to the team to boost the impact of design and design thinking and with it a better user experience for the developed products.

Within all those occasions the catalyst kit is a tool to start a discussion about processes, methods and tools used and to gain a common understanding on how to best integrate design within an interactive systems development team.

Part 5

Conclusions and future research

The objective of this research has been to encourage collaboration and design thinking in interactive systems development. Part 5 concludes this research and suggests future research.

Already some conclusions drawn from our experiences in many projects (see *Projects* (p 24) and *Teamwork: designers & engineers* (p 31)) have been summarized in the form of the catalyst kit (p 121). In the following I will elaborate on the position the catalyst kit has in regard to process design and its contribution to the interaction design community.

Using a holistic design approach as described in generation three (p 14) to product development is the basis for delivering a good user experience to the user as our projects have shown. But adopting such an approach is only possible by also questioning – or I prefer calling it *designing* – the process applied in developing the software. Both Lopp (2007) and Buxton (2007) argue that the design of the process is an important part in delivering a successful product:

> *The process is the product […] Like a good screw-up, a healthy argument provides a different perspective, which, if everyone is paying attention, will help the process evolve, and love it or hate it, the process is how you build a product. — Lopp 2007, p 72*

> *Innovation in process trumps innovation in product.*
>
> *In order to create successful products, it is as important (if not more) to invest in the design of the design process, as in the design of the product itself.*

[…] Innovation in process may trump innovation in product. But innovation in both trumps either. — Buxton 2007, p 408–9

Based on my experience of many projects the design of the process applied happens while working on the project itself. The methods, tools and techniques described in the catalyst kit are both an inspiration and a starting point for discussions – but the methods actually applied are altered for every single project. Important is what the catalysts trigger – a reflection of the methods used and the shaping of the process by the whole team – and with it the product.

Stolterman demands tools for interaction design practitioners, that support a designerly thinking, grounded in *a fundamental understanding of the nature of design practice* (Stolterman 2008). Buxton states that *we need to apply the same skills to the design of our tools, environment, and process as we do to the products that all these are intended to serve* (Buxton 2007, p 183). The catalyst kit is a contribution to Stolterman's request for tools for practitioners. With its open and flexible approach it gives room to tinker and to shape the toolkit as well as all its single entities of it to the needs of the actual project and environment and therefore supports Buxton's request for a designerly approach in designing the tools, the environment, and the process.

The catalyst kit is a collection of many of the methods and tools which helped us bringing design thinking into project teams to enable them to develop a better user experience. Not all of them worked equally good in all projects. But what always helped us were the discussions about the methods. I hope that the (catalyst) cards will trigger such discussions in other teams as well – and that they encourage collaboration and design thinking.

Future research

The development of the catalyst kit is based on my experience, mostly as an interaction designer, in several projects. It would be interesting to see catalyst cards developed by business people and engineers and their views on holistic product development.

Running an empirical study of the usage and impact of the catalyst kit on projects of different sizes and complexity would provide further insight on how to improve both its utilization and design.

Another appealing research area is connecting the catalyst kit to a digital system, so that the catalysts could provide easy reference to more elaborated methods and example documents while still being tangible and encourage discussion in a team session.

Bibliography

interactions. Whose profession is it anyway? (2005). Interactions, 12 (3). New York, NY, USA.

IXDA-mailing list: What sets the »best« interaction designers apart? (2007) <http://www.ixda.org/discuss.php?post=14357>

Merriam-Webster's online dictionary. (2008) <http://www.m-w.com/dictionary/>

37signals (2006) *Getting real: The smarter, faster, easier way to build a successful web application.* 37signals, LLC.

Alexander, C., Ishikawa, S., & Silverstein, M. (1978) *A Pattern Language: Towns, Buildings, Construction (Center for Environmental Structure Series).* Oxford University Press.

Anderson, R., Instone, K., Knemeyer, D., Mazur, B., & Quesenbery, W. (2005) *User experience network: A passion for collaboration.* Interactions, 12 (3), pp 40–41.

Arias, E. G., & Fischer, G. (2000) *Boundary Objects: Their Role in Articulating the Task at Hand and Making Information Relevant to It.* International ICSC Symposium on Interactive & Collaborative Computing (ICC 2000), pp 567–574.

Arnowitz, J., & Dykstra-Erickson, E. (2005a) *It's mine.* Interactions, 12 (3), pp 7–9.

Arnowitz, J., & Dykstra-Erickson, E. (2005b) *Usability as science.* Interactions, 12 (2), pp 7–8.

Arnowitz, J., Arent, M., & Berger, N. (2006) *Effective Prototyping for Software Makers (Interactive Technologies).* Morgan Kaufmann.

Ashley, J., & Desmond, K. (2005) *Success with user-centered design management.* Interactions, 12 (3), pp 27–32.

Bauer, F. (1993) *Software engineering – Wie es begann.* Informatik Spektrum, 16 (5), pp 259–260.

Beck, K., & Andres, C. (2004) *Extreme Programming Explained : Embrace Change.* 2nd Edition, Addison-Wesley Professional.

Beck, K., & Fowler, M. (2000) *Planning Extreme Programming (XP).* Addison-Wesley Longman, Amsterdam.

Beck, K., Beedle, M., Bennekum, A. V., Cockburn, A., Cunningham, W., Fowler, M., et al. (2001) *Manifesto for Agile Software Development.* <http://www.agilemanifesto.org/>

Belenguer, J., Parra, J., Torres, I., & Molina, P. J. (2003) *HCI Designers and Engineers: It is possible to work together?* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 14–19.

Berkun, S. (2005) *The Art of Project Management*. O' Reilly.

Bierut, M. (2005) *On (Design) Bullshit.* <http://www.designobserver.com/archives/entry.html?id=2559>

Bierut, M. (2006) *This is My Process.* <http://www.designobserver.com/archives/entry.html?id=17485>

Blevis, E., Lim, Y. K., & Stolterman, E. (2006) *Regarding Software as a Material of Design.* In: Wonderground 2006, Design Research Society International Conference 2006, Lisbon, Portugal.

Blevis, E., Lim, Y., Ozakca, M., & Aneja, S. (2005) *Designing interactivity for the specific context of designerly collaborations.* In: CHI '05 extended abstracts on Human factors in computing systems, pp 1216–1219.

Boehm, B. (2002). *Get Ready for Agile Methods, with Care.* Computer, 35(1), pp 64–69.

Boehm, B. (2006) *A view of 20th and 21st century software engineering.* ICSE '06: Proceeding of the 28th international conference on Software engineering, pp 12–29.

Boehm, B., & Turner, R. (2003) *Using Risk to Balance Agile and Plan-Driven Methods.* Computer, 36 (6), pp 57–66.

Bogaards, P., & Priester, R. (2005) *User experience: back to business.* Interactions, 12 (3), pp 23–25.

Borchers, J. (2001) *A Pattern Approach to Interaction Design*. John Wiley & Sons.

Brooks, F. P. (1987) *No Silver Bullet Essence and Accidents of Software Engineering.* Computer, 20 (4), pp 10–19.

Brown, T. (2005) *Strategy by Design.* Fast Company (95). <http://www.fastcompany.com/magazine/95/design-strategy.html>

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

Buxton, W. (2005) *Innovation vs. Invention.* Rotman Magazine, The Magazine of the Rotman School of Management, Fall 2005, pp 52–54.

Carlile, P. R. (2002) *A Pragmatic View of Knowledge and Boundaries: Boundary Objects in New Product Development.* Organization Science, 13 (4), pp 442–455.

Carlile, P. R. (2004) *Transferring, Translating, and Transforming: An Integrative Framework for Managing Knowledge Across Boundaries.* Organization Science, 15 (5), pp 555–568.

Carroll, J. M. (2000) *Making Use: Scenario-Based Design of Human-Computer Interactions*. The MIT Press.

Chamberlain, S., Sharp, H., & Maiden, N. (2006) *Towards a Framework for Integrating Agile Development and User-Centred Design.* Extreme Programming and Agile Processes in Software Engineering, pp 143–153.

Clemmensen, T., & Nørbjerg, J. (2003) *Separation in Theory - Coordination in Practice.* ICSE Workshop on SE-HCI, pp 100–105.

Cohn, M. (2005) *Agile Estimating and Planning*. Prentice Hall PTR.

Constantine, L. L., Biddle, R., & Noble, J. (2003) *Usage-Centered Design and Software Engineering: Models for Integration.* ICSE Workshop on SE-HCI, pp 106–113.

Cooper, A. (1999) *The inmates are running the asylum : Why high tech products drive us crazy and how to restore the sanity.* 1st Edition, Sams.

Cooper, A. (2004) *The inmates are running the asylum : Why high tech products drive us crazy and how to restore the sanity.* 2nd Edition, Sams.

Cooper, A., & Reimann, R. M. (2003) *About Face 2.0: The Essentials of Interaction Design*. Wiley.

Darke, J. (1978) *The primary generator and the design process.* In: Design methods in architecture. Lund Humphries, London,

DeGrace, P., & Hulet-Stahl, L. (1990) *Wicked Problems, Righteous Solutions : A Catalogue of Modern Software Engineering Paradigms*. Yourdon Press.

Dewey, J. (1999) *The Quest for Certainty: a study of the relation of knowledge and action*. Minton Balch, New York.

Dix, A., Finlay, J. E., Abowd, G. D., & Beale, R. (2003) *Human-Computer Interaction.* 3rd Edition, Prentice Hall.

Doctorow, C. (2008) *Upload*. Heyne.

van Duyne, D. K., Landay, J. A., & Hong, J. I. (2006) *The Design of Sites: Patterns for Creating Winning Web Sites.* 2nd Edition, Prentice Hall PTR.

Erickson, T., & Thomas, J. (1997) *Putting it all together: pattern languages for interaction design.* In: CHI '97 extended abstracts on Human factors in computing systems.

Ferré, X. (2003) *Integration of Usability Techniques into the Software Development Process.* ICSE Workshop on SE-HCI, pp 28–35.

Ferreira, J., Noble, J., & Biddle, R. (2007) *Up-Front Interaction Design in Agile Development.* In: Agile Processes in Software Engineering and Extreme Programming, pp 9–16.

Fischer, G. (2001) *Communities of Interest: Learning through the Interaction of Multiple Knowledge Systems.* Proceedings of the 24th IRIS Conference, pp 1–14.

Floyd, C. (1992a) *Human Questions in Computer Science.* In: C. Floyd, H. Züllighoven, R. Budde, & R. Keil-Slawik (Eds.), Software Development as Reality Construction. Springer Verlag. Springer Verlag, Berlin, pp 15–27.

Floyd, C. (1992b) *Software Development as Reality Construction.* In: C. Floyd, H. Züllighoven, R. Budde, & R. Keil-Slawik (Eds.), Software Development as Reality Construction. Springer Verlag, Berlin, pp 86–100.

Folmer, E., & Bosch, J. (2004) *Cost effective development of usable systems; gaps between HCI and SE.* IEE Seminar Digests, 2004 (901), pp 40–45.

Folmer, E., Gurp, J. V., & Bosch, J. (2003) *Scenario-based Assessment of Software Architecture Usability.* ICSE Workshop on SE-HCI, pp 61–68.

Folmer, E., Welie, M., & Bosch, J. (2006) *Bridging patterns: An approach to bridge gaps between SE and HCI.* Information and Software Technology, 48 (2), pp 69–89.

Fortner, R. (2007) *Software project management in unstable environments - handling volatile business requirements induced by trade-offs among client stakeholders*. Masters Thesis, Vienna Technical University, Vienna.

Frankfurt, H. G. (2005) *On Bullshit.* Princeton University Press.

Fried, J. (2007) *Ask 37signals: Personas?* Signal vs. Noise. <http://www.37signals.com/svn/posts/690-ask-37signals-personas>

Fried, J. (2008) *Why we skip Photoshop.* Signal vs. Noise. <http://www.37signals.com/svn/posts/1061-why-we-skip-photoshop>

Gabriel-Petit, P. (2005) *Introduction: sharing ownership of UX.* Interactions, 12 (3), pp 16–18.

Gasson, S. (2005) *The dynamics of sensemaking, knowledge, and expertise in collaborative, boundary-spanning design.* Journal of Computer-Mediated Communication, 10 (4),

Gedenryd, H. (1998) *How designers work*. PhD dissertation, Cognitive Studies Department, Lund University, Sweden.

Gellner, M., & Forbrig, P. (2003) *Extreme Evaluations – Lightweight Evaluations for Software Developers.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 75–80.

Go, K., & Carroll, J. M. (2004) *The blind men and the elephant: views of scenario-based system design.* Interactions, 11 (6), pp 44–53.

Goodman, B. (2005) *Making UX an engaging process for prospective user experience adopters.* Interactions, 12 (3), pp 25–26.

Graham, I. (2003) *A Pattern Language for Web Usability*. Pearson Education.

Grudin, J. (2006) *Is HCI homeless?: in search of inter-disciplinary status.* Interactions, 13 (1), pp 54–59.

Hansson, D. H. (2008) *Web designers should do their own HTML/CSS.* Signal vs. Noise. <http://www.37signals.com/svn/posts/1066-web-designers-should-do-their-own-htmlcss>

Harning, M. B., & Vanderdonckt, J., Eds. (2003) *Closing the Gaps: Software Engineering and Human-Computer Interaction*. Institut d' Administration et de Gestion (IAG), Université catholique de Louvain (UCL).

Harning, M. B., Kazman, R., Kohler, K., Kerkow, D. , & Gulliksen, J., Eds. (2005) *Integrating Software Engineering and Usability Engineering.* INTERACT 2005, Rome, Italy.

Hawdale, D. (2005) *The vision of good user experience.* Interactions, 12 (3), pp 22–23.

Helander (1998) *Handbook of Human-Computer Interaction.* 2nd Edition, Elsevier Science Pub Co.

Hoffmann, J. (2005). *Adaptive Page Layout.* <http://www.robocup.de/aiboteamhumboldt/team/adaptivelayout.html>

Horner, J., & Atwood, M. E. (2006) *Design rationale: The rationale and the barriers.* In: NordiCHI '06 proceedings of the 4th Nordic conference on Human-computer interaction, pp 341–350.

IDEO (2003) *IDEO method cards: 51 ways to inspire design*.

Ionita, M. T., America, P., Obbink, H., & Hammer, D. (2003) *Quantitative Architecture Usability Assessment with Scenarios.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 27–33.

Jacobson, I., Booch, G., & Rumbaugh, J. (1999) *The Unified Software Development Process.* Addison-Wesley Professional.

Jeffries, R., Turner, A., Polson, P., & Atwood, M. (1981) *The process involved in designing software.* In: Cognitive Skills and Their Acquisition. Lawrence Erlbaum, pp 255–83.

Jespersen, J. W., & Linvald, J. (2003) *Investigating User Interface Engineering in the Model Driven Architecture.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 63–66.

John, B. E., Bass, L., Kazman, R., & Chen, E. (2004) *Identifying gaps between HCI, software engineering, and design, and boundary objects to bridge them.* In: CHI '04 extended abstracts on Human factors in computing systems, pp 1723–1724.

Jones, S., Maiden, N. A. M., Manning, S., & Greenwood, J. (2004) *Human activity modelling in the specification of operational requirements: work in progress.* IEE Seminar Digests, 2004 (901), pp 1–8.

Jörgensen, K. (1992) *Scientific Work Paradigms.* Technical report. Institute for Production, Aalborg University, Denmark.

Juzgado, N. J., López, M., Moreno, A., & Segura, M. I. S. (2003) *Improving software usability through architectural patterns.* ICSE Workshop on SE-HCI, pp 12–19.

Kalbach, J. (2002) *Challenging the Status Quo: Audi Redesigned.* boxes and arrows. <http://www.boxesandarrows.com/view/challenging_the_status_quo_audi_redesigned>

Kalbach, J., & Bosenick, T. (2006) *Web Page Layout: A Comparison Between Left- and Right-justified Site Navigation Menus.* Journal of Digital Information, 4 (1),

Karlsson, A. (2002) *Developing High Performance Manufacturing Systems.* PhD dissertation, Department of Production Engineering, Royal Institute of Technology, Stockholm, Sweden.

Kazman, R., Bass L., & Bosch J., Eds. (2003) *Bridging the Gaps Between Software Engineering and Human-Computer Interaction.* International Conference on Software Engineering 2003, Portland, Oregon, USA.

Kazman, R., Bass, L., & John, B., Eds. (2004) *Bridging the Gaps II: Bridging the Gaps Between Software Engineering and Human-Computer Interaction*. International Conference on Software Engineering 2004, Edinburgh, Scotland.

Kerkow, D., Schmidt, K., & Wiebelt, F. (2005) *Requirements for the Integration of UE Methods in SE Processes from the Perspective of Small and Medium-sized Enterprises (SMEs)*. In: Integrating Software Engineering and Usability Engineering.

Kirsh, D., & Maglio, P. P. (1992) *Some epistemic benefits of action: Tetris, a case study.* In: Proceedings of the Fourteenth Annual Conference of the Cognitive Science Society.

Knemeyer, D. (2005) *Who owns UX?: Not us!.* Interactions, 12 (3), pp 18–20.

Kovalchuke, O. (2006) *Interaction Design Process and Tools.* <http://www.tangospring.com/IxDtopicProcessTools.htm>

Laakso, S. A. (2003) *User Interface Design Patterns.* <http://www.cs.helsinki.fi/u/salaakso/patterns/index.html>

Lash, J., & Baum, C. (2007a) *Transitioning from User Experience to Product Management (Part 1)*. boxes and arrows. <http://www.boxesandarrows.com/view/transitioning-from>

Lash, J., & Baum, C. (2007b) *Transitioning from User Experience to Product Management (Part 2)*. boxes and arrows. <http://www.boxesandarrows.com/view/transitioning-from26>

Lave, J., & Wenger, E. (1991) *Situated Learning : Legitimate Peripheral Participation (Learning in Doing: Social, Cognitive & Computational Perspectives).* Cambridge University Press.

Law, E. L. (2003) *Bridging the HCI-SE Gap: Historical and Epistemological Perspectives.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 47–54.

Lawson, B. (1997) *How Designers Think, Third Edition: The Design Process Demystified*. First published 1980, completely revised 3rd Edition 1997, Oxford: Butterworth Architectural Press, 1998.

Leffingwell, D., & Widrig, D. (1999) *Managing Software Requirements: A Unified Approach*. Addison-Wesley Professional.

Lopp, M. (2007) *Managing Humans: Biting and Humorous Tales of a Software Engineering Manager*. Apress.

Louridas, P., & Loucopoulos, P. (2000) *A generic model for reflective design.* ACM Trans. Softw. Eng. Methodol., 9 (2), pp 199–237.

Lovegrove, R. (2007) *Interview with Ross Lovegrove*. In: lebens.art on October 8, 2007. ORF.

Löwgren, J. (1995) *Applying design methodology to software development.* DIS '95: Proceedings of the 1st conference on Designing interactive systems, pp 87–95.

Löwgren, J., & Purgathofer, P. (2006) *sketching for interactions.* <http://twoday.tuwien.ac.at/interactionsketching/>

Ludi, S. (2003) *Undergraduate Software Engineering Curriculum Enhancement via Human-Computer Interaction.* ICSE Workshop on SE-HCI, pp 72–75.

Mahanti, A. (2006) *Challenges in Enterprise Adoption of Agile Methods – A Survey.* Journal of Computing and Information Technology (CIT), 14 (3), pp 197–206.

Marick, B. (2003) *Boundary objects*. Analogy Fest. Agile Development Conference, June 25-28, 2003.

McPhee, K. (1996) *Design Theory and Software Design.* Technical Report TR 96-26. Department of Computer Science, University of Alberta, Edmonton, Canada.

Merholz, P. (2007) *Interview with Jess McMullin: Smoothing the Way: Designer as Facilitator.* <http://www.adaptivepath.com/blog/2007/07/13/interview-with-jess-mcmullin/>

Meyer, B. (1989) *From Structured Programming to Object-Oriented Design: The Road to Eiffel.* Structured Programming, 10 (1), pp 19–39.

Microsoft (2008a). *Microsoft® Expression®* (Computer software).

Microsoft (2008b) *XAML Overview.* <http://msdn.microsoft.com/en-us/library/ms752059.aspx>

Milewski, A. E. (2003) *Software Engineering Overlaps with Human-Computer Interaction: A Natural Evolution.* ICSE Workshop on SE-HCI, pp 69–71.

Miller, L. (2005) *Case Study of Customer Input For a Successful Product.* ADC '05: Proceedings of the Agile Development Conference, pp 225–234.

Moggridge, B. (2006) *Designing Interactions*. The MIT Press.

Moran, T. P., & Carroll, J. M., Eds. (1996) *Design Rationale: Concepts, Techniques, and Use (Computers, Cognition, and Work).* CRC.

Müller-Prove, M. (2003) *Mind the Gap! Software Engineers and Interaction Architects.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 100–101.

Nelson, E. (2002) *Extreme Programming vs. Interaction Design.* Original source offline, access via The Wayback Machine: <http://web.archive.org/web/20070313205440/http://www.fawcette.com/interviews/beck_cooper/>

Norman, D. A. (2004a) *Ad-Hoc Personas & Empathetic Focus.* <http://www.jnd.org/dn.mss/personas_empath.html>

Norman, D. A. (2004b) *Emotional Design: Why We Love (Or Hate) Everyday Things*. Basic Books.

Norman, D. A. (2005) *Whose profession is this?: everybody's, nobody's.* Interactions, 12 (3),

Norman, D., & Draper, S. (1986) *User Centered System Design: New Perspectives on Human-computer Interaction*. TF-CRC.

Nunes, N. J. (2003) *What drives software development: issues integrating software engineering and human-computer interaction.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 89–95.

Paech, B., & Kohler, K. (2003) *Usability Engineering integrated with Requirements Engineering.* ICSE Workshop on SE-HCI, pp 36–40.

Palanque, P., & Bastide, R. (2003) *UML for Interactive Systems: What is Missing.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 96–99.

Parnas, D. L., & Clements, P. C. (1986) *A rational design process: How and why to fake it.* IEEE Trans. Softw. Eng., 12 (2), pp 251–257.

Patton, J. (2006) *The Whole Product*. <http://www.stickyminds.com/s.asp?F=S11662_COL_2>

Pavese, A. (2007) *Get real: How to design for the life of others?* <http://www.antonellapavese.com/archive/2007/04/249/>

Pichlmair, M. (2004) *Designing for Emotions – arguments for an emphasis on affect in design.* PhD dissertation, Design and Assessment of Technology Institute, Vienna Technical University, Vienna.

Porter, J. (2007) *Five Principles to Design By.* <http://bokardo.com/archives/five-principles-to-design-by/>

Portigal, S. (2008) *Persona non grata.* Interactions, 15 (1), pp 72–73.

Preece, J., Rogers, Y., & Sharp, H. (2002) *Interaction Design: Beyond Human Computer Interaction*. Wiley.

Pruitt, J., & Adlin, T. (2006) *The The Persona Lifecycle: Keeping People in Mind Throughout Product Design.* 1st Edition, Morgan Kaufmann.

Purgathofer, P. (2003) *Designlehren: Zur gestaltung interaktiver systeme*. Habilitation treatise, Design and Assessment of Technology Institute, Vienna Technical University, Vienna.

Purgathofer, P. (2006) *Is informatics a design discipline?* Poiesis & Praxis: International Journal of Technology Assessment and Ethics of Science, 4 (4), pp 303–314.

Quesenbery, W., Anderson, R., & Mazur, B. (2005). *UXnet: making connections.* In: CHI '05 extended abstracts on Human factors in computing systems 1098–1099.

Rittel, H. W. J., & Webber, M. M. (1973) *Dilemmas in a general theory of planning.* Policy Sciences, 4 (2), pp 155–169.

Rosenberg, D. (2006) *Revisiting tangible speculation: 20 years of UI prototyping.* Interactions, 13 (1), pp 31–32.

Saffer, D. (2006) *Designing for Interaction: Creating Smart Applications and Clever Devices.* Peachpit Press.

Sherman, P. J. (2007) *Envisioning the Future of User Experience.* <http://www.uxmatters.com/MT/archives/000184.php>

Schmettow, M. (2005) *Towards a pattern based usability inspection method for industrial practitioners.* In: Integrating Software Engineering and Usability Engineering.

Schön, D. A. (1983) *The Reflective Practitioner: How Professionals Think in Action*. Basic Books.

Schwaber, K., & Beedle, M. (2001) *Agile Software Development with Scrum*. Prentice Hall PTR.

Seffah, A. (2004) *Bridging the educational gap between SE and HCI: What software engineers should know?* IEE Seminar Digests, 2004 (901), pp 88–95.

Sherman, P., & Quesenbery, W. (2005) *Engineering the user experience: UX and the Usability Professionals' Association.* Interactions, 12 (3), pp 38–40.

Shneiderman, B., & Plaisant, C. (2004) *Designing the User Interface : Strategies for Effective Human-Computer Interaction.* 4th Edition. Addison Wesley.

Silva, P. P. D., & Paton, N. W. (2003) *Improving UML Support for User Interface Design: A Metric Assessment of UMLi.* ICSE Workshop on SE-HCI, pp 76–83.

Simon, H. A. (1996) *The Sciences of the Artificial.* 3rd Edition, The MIT Press.

Snow, K., Marks, M., Hong, D., & Dennis, T. (2006). *Web Patterns.* <http://groups.ischool.berkeley.edu/ui_designpatterns/webpatterns2/webpatterns/home.php>

Sommerville, I. (2006) *Software Engineering.* 8th Edition, Addison Wesley.

Sousa, K. S., & Furtado, E. (2003a) *An Approach to Integrate HCI and SE in Requirements Engineering.* In: Closing the Gaps: Software Engineering and Human-Computer Interaction, pp 81–88.

Sousa, K. S., & Furtado, E. (2003b) *RUPi - A Unified Process that Integrates Human-Computer Interaction and Software Engineering.* ICSE Workshop on SE-HCI, pp 41–48.

Sousa, K. S., & Furtado, E. (2004) *UPi - a unified process for designing multiple UIs.* IEE Seminar Digests, 2004 (901), pp 46–53.

Spangl, J., & Haberfellner, T., Eds. (2004) *Interaction Design Stammtisch Wien*. <http://www.interactiondesign.at/>

Spool, J. (2007a) *Crappy Personas vs. Robust Personas.* <http://www.uie.com/brainsparks/2007/11/14/crappy-personas-vs-robust-personas/>

Spool, J. (2007b) *We Are Not Alone: IA's Role in the Optimal Design Team*. (Podcast) <http://www.uie.com/brainsparks/2006/03/28/ia-summit-presentation-we-are-not-alone/>

Star, S. L. (1990) *The structure of ill-structured solutions: Boundary objects and heterogeneous distributed problem solving.* In: Distributed artificial intelligence. Morgan Kaufmann Publishers Inc., pp 37–54.

Stolterman, E. (2008) *The Nature of Design Practice and Implications for Interaction Design Research.* International Journal of Design, 2 (1), pp 55–65.

Taylor, R. N., & van der Hoek, A. (2007) *Software Design and Architecture The once and future focus of software engineering.* FOSE '07: 2007 Future of Software Engineering, pp 226–243.

Tidwell, J. (2005a) *Designing Interfaces : Patterns for Effective Interaction Design*. O' Reilly Media, Inc.

Tidwell, J. (2005b). *Designing Interfaces : Patterns for Effective Interaction Design.* <http://designinginterfaces.com/>

Tognazzini, B. (2005) *Why engineers own user experience design.* Interactions, 12 (3), pp 32–34.

Toxboe, A. (2008) *User Interface Design Patterns.* <http://ui-patterns.com/>

Vanka, S. (2007) *Microsoft Expression Designer Tour*. Vienna, October 8, 2007.

Virzi, R. A., Sokolov, J. L., & Karis, D. (1996) *Usability problem identification using both low- and high-fidelity prototypes.* In: CHI '96 proceedings of the SIGCHI conference on Human factors in computing systems, pp 236–243.

Walters, H. (2008) *Apple's design process.* Businessweek. <http://www.businessweek.com/the_thread/techbeat/archives/2008/03/apples_design_p.html>

van Welie, M. (2001) *Task-based User Interface Design*. PhD dissertation, Vrije Universiteit Amsterdam.

van Welie, M. (2007). *Patterns in Interaction Design.* <http://www.welie.com/>

van Welie, M., & van der Veer, G. C. (2003) *Pattern Languages in Interaction Design: Structure and Organization.* Human-Computer Interaction – INTERACT '03, pp 527–434.

Wilkinson, N. M. (1995) *Using CRC cards: An informal approach to object-oriented development*. SIGS Publications, Inc.

Willshire, M. J. (2003) *Where SE and HCI Meet: A Position Paper.* ICSE Workshop on SE-HCI, pp 57–60.

Winograd, T. (1995) *From programming environments to environments for designing.* Commun. ACM, 38 (6), pp 65–74.

Winograd, T., Bennett, J., Laura, & Hartfield, B. (1996) *Bringing Design to Software*. Addison-Wesley Professional.

Wolf, T. V., Rode, J. A., Sussman, J., & Kellogg, W. A. (2006) *Dispelling »design« as the black art of CHI.* In: CHI '06 proceedings of the SIGCHI conference on Human Factors in computing systems, pp 521–530.

Yahoo! (2008). *Design Pattern Library.* <http://developer.yahoo.com/ypatterns/index.php>

Zeldman, J., & Lopp, M. (2008) *Studio SX Presents Jeffery Zeldman and Michael Lopp*. (Video) <http://video.sxsw.com/2008/mov/lopp_zeldman.mov>

Appendix

The catalyst cards

The catalyst cards are provided in two versions:

- **One-sided:** This is the best option if you can only print on paper of weight less than 200 g/m². Folding them, and adding some glue improves the stability. The one-sided version is shown on the following pages.
- **Two-sided:** The cards are aligned for easy print out. Best used with paper of weight equal or greater than 200 g/m².

Both versions of the catalyst cards and templates for *your* cards can be found online:

- http://nuup.com/catalystkit/

Process

Question the process

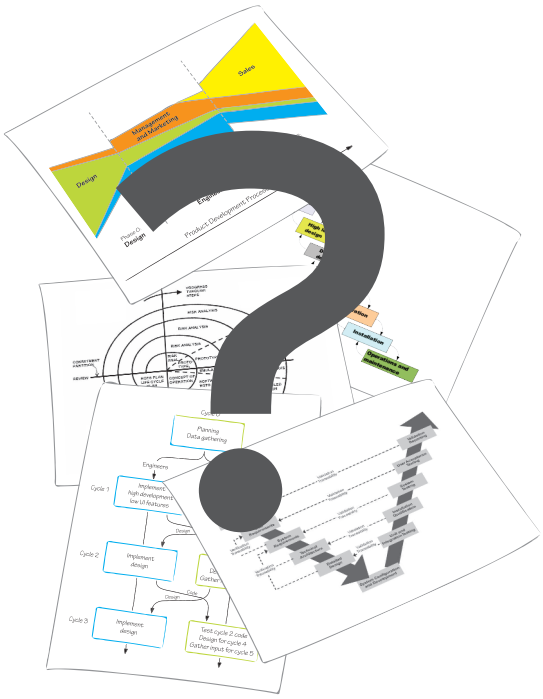
How
Question the process: Does the currently followed process still make sense? Are all viewpoints included? Does the process support a holistic approach? Are business, design, and engineering equally represented? Do all team members in the project fully understand and comply with the process?

Why
A good process should be open for change. It should provide mechanisms to constantly review itself. Designing the process is also a part of design. Team members need to feel responsible for the results and the impacts processes have on projects, therefore they need to fully understand and support the process.

→ Summary – Process (p 45)

Process

Question the process



Process

Combine processes

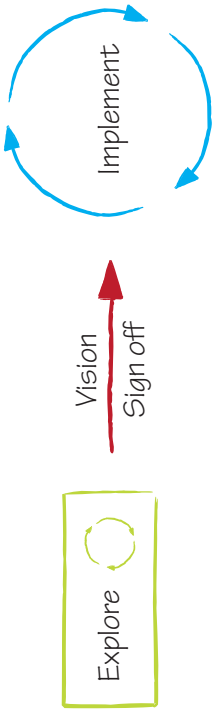
How
Combine elements from plan-driven and agile environments. Start with a user-centered phase with user research and explore different concepts up-front before starting implementation. This could even include building some prototypes iteratively to get a feeling for the product. Once agreement is established for an overall concept (vision) it builds the basis for the user stories. The implementation phase then uses an agile approach. Remaining design issues are clarified and detailed during the iterations. Additionally, the quality of the design is reviewed and evaluated with usability tests.

Why
Working out the concepts up-front provides the whole team a vision for the product. Furthermore design isn't limited by the technical implementation, and hereby faster and can factor in user feedback earlier. Compared to plan-driven environments the combination gives more flexibility in responding to changing requirements and to findings from the usability tests.

→ Up-front design (p 36)
→ Our experience with processes (p 41)

Process

Combine processes



Process

n±1

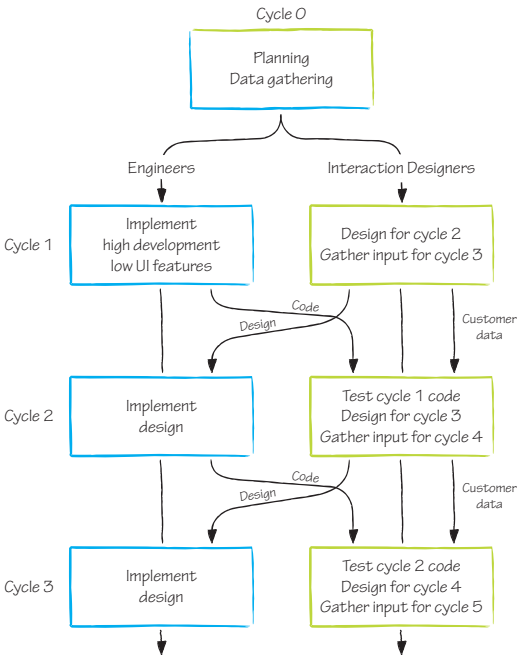
How
How to better include design in agile environments? Designers should support the product owner (customer) by describing and choosing the user stories for the next iteration, based on the findings from user research. The design for the features should be finished before the iteration for implementation starts – one iteration ahead (n-1). The user stories get implemented in iteration n. The following iteration (n+1) is used for usability testing and refinement of the features, just implemented. Hence the designers will – in every iteration – design and detail the user stories planned for the next iteration, and conduct usability tests and refine the user stories implemented in the last iteration.

Why
n±1 involves not only the customer, but also the user. It interprets the user needs into features and saves time by combining the user research and usability test into one session with the user. Engineers can focus on the technical aspects, without loosing time by waiting for the designers who are working in the same iteration on the same user story.

→ Integrated iterative interaction design (p 38)

Process

n±1



Source: Miller, 2005

Process

Stay in the game

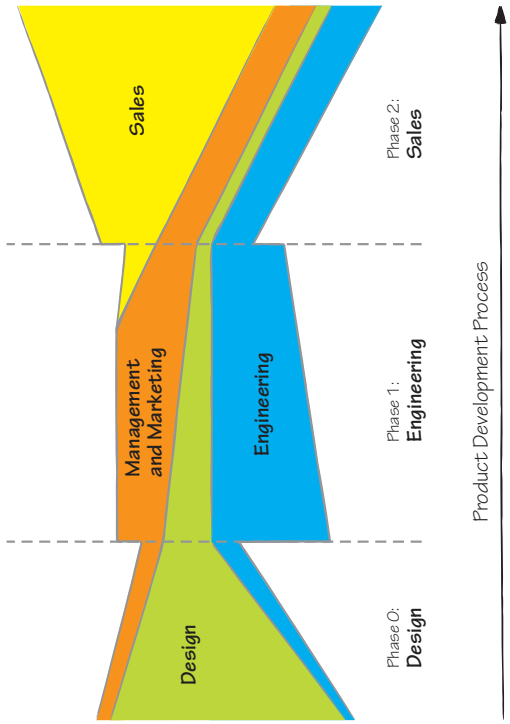
How
All parties – business people, designers, and engineers – should be involved during the whole product development life cycle. Depending on the phase of the project, the involvement of each of the parties might differ – at the beginning it usually leans more towards design, later on engineering takes over.

Why
Engineers are needed from the very beginning on to provide technical constraints and to check the technical feasibility of the design concepts. During later phases the involvement of the designers is important to assure that the implementation conforms to the concepts and provide designs of upcoming issues.

→ Design in plan-driven environments (p 33)
→ Our experience with processes (p 41)

Process

Stay in the game



Source: Buxton, 2007

Process

Pair design/programming

How
Pair design/programming is a reference to pair programming in extreme programming. Instead of two programmers pairing up, a designer and a programmer sit together in front of the computer and tinker with certain aspects of the user interface and behavior of the application. This is usually done on a branch of the actual product and without adhering to programming guidelines. It's a form of prototyping done on a branch of the product.
Probably most useful in agile environments, but it can also be applied within plan-driven environments during the implementation phase to decide arising detailed issues.

Why
Working with the actual material is often needed to decide on certain design details. Building prototypes for every detail is too time consuming and expensive, and often design issues not arise until implementation.

→ Our experience with prototypes (p 95)

Process

Pair design/programming

Collaboration

Power ratio = 1:1:1

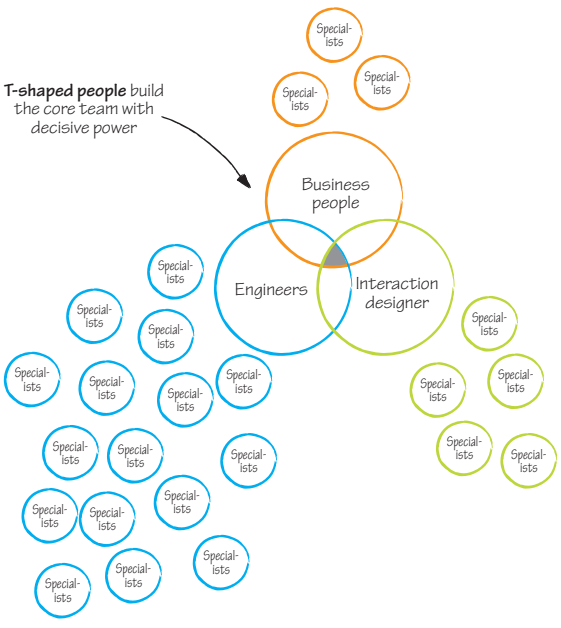
How
Balance the power within the team 1:1:1 among the involved perspectives: business to technology to user. The power is democratic within the three perspectives and not the whole team.
The balance can be established by a core team, consisting of a member of each perspective. Ideally the core team would be staffed with T-shaped people – people who have a core competency, but can easily branch out into other skills.

Why
The interdisciplinary view of the project team is crucial for developing a product which delivers a good user experience. Every perspective counts equally. Over the course of the project the power might shift between the perspectives, e.g. at the beginning the power of the user perspective is higher, during implementation this may change towards technology. But still all parties are involved in the decisions. Over the lifespan of the whole project the power ratio is balanced.

→ Balanced teams (p 68)

Collaboration

Power ratio = 1:1:1



Collaboration

The catalyst role

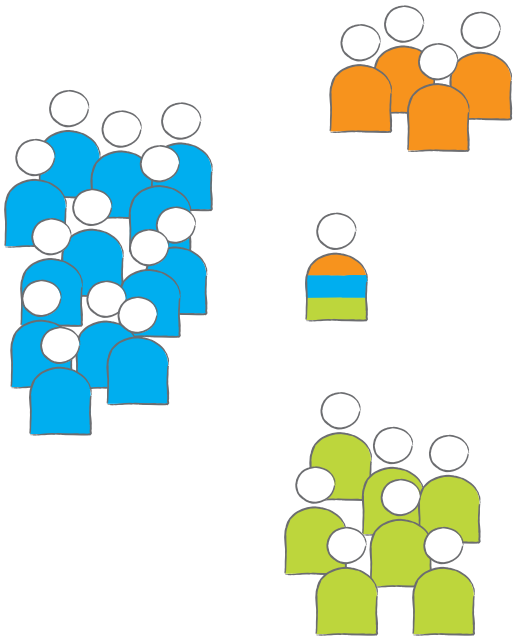
How
Facilitating consensus among the different viewpoints is the main task of the catalyst role. The individuals who take this role have to be able to understand the different languages of all perspectives involved. People of the core team need to have the capabilities of the catalyst role. On larger teams the catalyst role could even be dedicated to a single person and be this persons single role.
Often the designers end up or take on this catalyst role, because their skills in observing, listening, synthesis, and in creating artifacts that people can relate to are also relevant to facilitate consensus.

Why
Delivering a good user experience is the responsibility of everyone in a project team and calls for the input of all disciplines involved. Synthesizing the divergent languages and often contrasting viewpoints of all disciplines is key to satisfy this demand.

→ The catalyst role (p 67)
→ Capabilities of an interaction designer (p 52)

Collaboration

The catalyst role



Collaboration

Switch roles

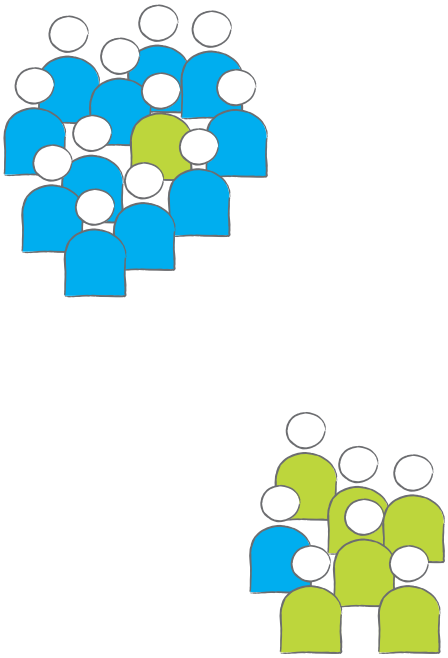
How
Switching roles over a short time frame together with working along your team members from a different discipline helps exchanging knowledge and learning each others' methods and tools. This is an excellent tool to gain the capabilities needed for the catalyst role.

Why
Understanding each others' languages, methods and techniques is the basis for finding consensus rather than compromise. Only consensus will deliver a product with good overall user experience.

→ The catalyst role (p 67)

Collaboration

Switch roles



Collaboration

The user experience is your responsibility

How
Take responsibility for the results and impacts processes have, for your decisions and artifacts in a project. For designers this means taking responsibility for their concepts and staying in the project until they reach the user. For engineers this means taking responsibility to get involved early in the project to inform the design team of constraints and support them by checking the technical feasibility early on. Furthermore it's their responsibility to implement the product as close as possible to the concepts.
Acting in a designerly way is the responsibility of everyone, but bringing design thinking to the team is often the responsibility of the designers. Take the responsibility, for a better user experience.

Why
Who owns user experience? – The team.
A good user experience can only be delivered if everyone within the team contributes his share. Hence everyone should accept and carry the responsibility for the user experience of the product.

→ Who owns user experience? (p 49)

Collaboration

The user experience is your responsibility

Collaboration

Kill your darlings – or at least question them

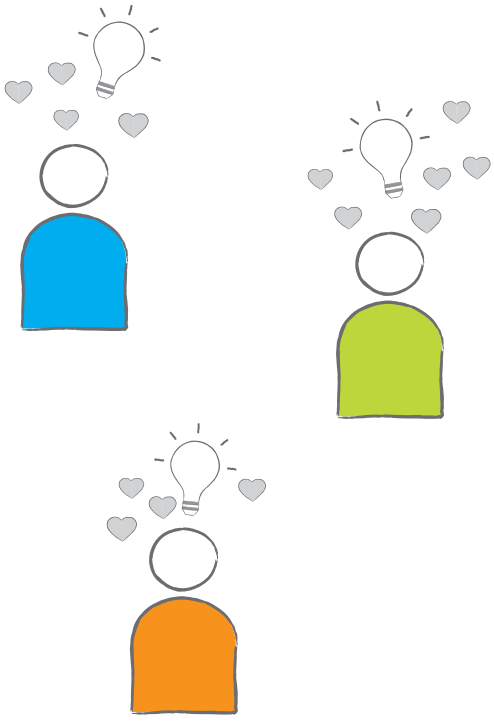
How
Do not fall in love with your ideas. Encourage others to question them. Try to get to the bottom of them.
On the other hand: fight for your concepts. Don't throw them over board too quickly, because you get the feedback that they're technically not feasible – question also such a feedback. But be honest with yourself: are you fighting for your idea because it's your idea – or because it's in the interest of the user?

Why
Getting emotionally too attached to your concepts, especially your first ones, can block other ideas and thus can put the project at risk. On the contrary, just following force of habit and using the same solutions over and over without questioning them will never allow for innovation, no more than using concepts only because the development environment provides them.

→ Generation three: doing for the sake of knowing (p 14)

Collaboration

Kill your darlings – or at least question them



Collaboration

Give reason

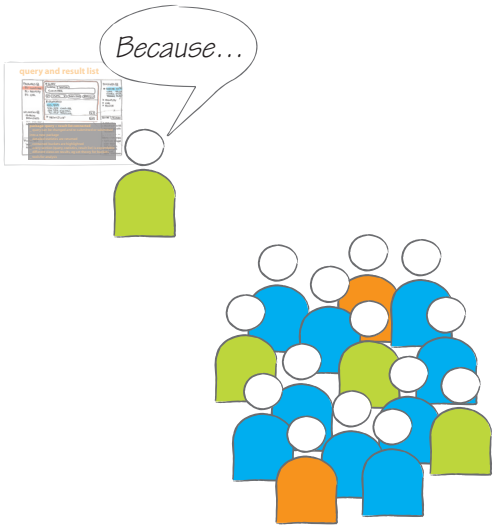
How
Communicate your judgements and your design rationale to your recipients. Describe your drivers, motivations and reasons during the design process. Also explain how you are doing it. Let others take part in your methods and approaches.

Why
Giving reason helps the team to build up a common ground in understanding each others' needs to be able to provide valuable feedback and input to the ideas and concepts. Furthermore it transports the approach of design thinking.

→ Getting your point across (p 55)

Collaboration

Give reason



Collaboration

Design your artifacts for the recipients

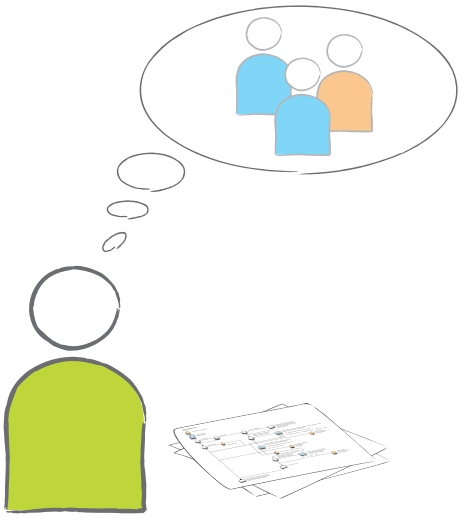
How
The artifacts should always transport their intention: be it for gathering feedback (e.g. prototypes) or to communicate decisions (e.g. functional specifications).
Prepare the deliverable for the recipients as you would design the product for the users. This also transports the quality of the design. If designers produce high quality deliverables, also the engineers and the rest of the team will create high quality deliverables, and this will result in a high quality product.
Take responsibility for your artifacts, especially for the specifications. Someone has to decide on the details – if the designers aren't doing it then the engineers have to do it, otherwise they cannot implement it.

Why
Selling the concept to the client is an important part of design. But this is also true for selling it to other team members. Selling doesn't only cover the presentation, but must be extended to the way your artifacts are edited.

→ Getting your point across (p 55)

Collaboration

Design your artifacts for the recipients



Collaboration

Specifications – clarify them

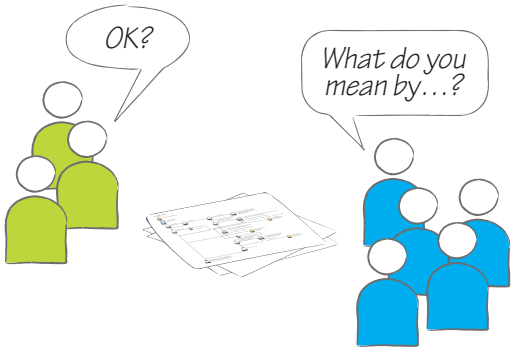
How
Specifications summarize the decisions made during the design process and define the product in different ways (see the artifact cards) in an implementation friendly style.
As such it's important to gain an understanding of the engineers on which parts need to be captured in which level of detail. In this regard it's useful to compile a small part of the specifications and then discuss them with the engineers – test your specifications with the engineers (the users in this case) as you would test your designs with the user.
In agile environments the specifications are usually on a per feature basis and not as detailed as in plan-driven environments. They're elaborated user stories including detailed screens and flows. But again, the level of detail has to be agreed upon within the team.

Why
It's in the interest of the team – but in the responsibility of the designers – to shape the transformation of the design into implementation as smoothly as possible.

→ Specifications (p 103)

Collaboration

Specifications – clarify them



Artifacts

Personas

How
Make your users part of the team by using personas. Bring them to every meeting and share them among the team members.

Personas:
› are a portrait of a typical user ideally based on user research data as well as input from the client.
› are hypothetical.
› are archetypical and not stereotypical.
› represent important demographic data.
› live in a social context.
› have characteristics and goals.
› are *alive*.

Why
Personas are a tool to give the anonymous users a face and to encourage team members to distinguish and also articulate their own ideas apart from the user's view.
In plan-driven environments personas can stand in as actors for use case models. In agile environments personas take the user role within the user stories.

→ Personas (p 73)

Artifacts

Personas

hans-dieter strunke, 46
primary user

the meister

meister and head of the department »prosthetics lower extremities« at sanitätshaus monke in stuttgart, germany

»da paßt noch nicht alles«

hans-dieter is one of four meister in his branch. his day is split between consulting patients and doing the paper work — which is ever increasing by the day. in the patient treatment he does the consulting, makes sure that the socket fits and selects the right components for the prosthesis. usually he orders the test-socket at otto bock, building the definite socket and the prosthesis is done by his coworkers. for the administrative work he is using the computer more and more, especially for patient management.

hans-dieter's goals
- i want to make sure that everything is right, but i don't want to spend too much time in front of the computer
- my workflow works best for me, i don't want to change it just to use the computer
- once i hand over the job to the assistant i want to be sure he gets everything

company information
- provides services in prosthetics, rehab & ortho shoes
- 60 employees, 3 branches
- he works in the head office with about 40 employees
- the 2 branches are smaller, each with 8 and 12 employees respectively

psychographics
- late mainstream — skeptical, adopting only after a majority have done so
- he only jumps onto new things if he really sees that it's worth it

computer proficiency and usage
- beginner
- daily uses sanivision for patient management
- infrequently excel for calculations
- daily email but only gets about 3–4 emails/week
- digital camera for patient documentation
- at home he surfs the internet infrequently on his son's pc, mainly for information about traveling, he also got an internet account from his bank but does not use it

computer equipment used by him
- pc for himself, about 1 1/2 years old, with 17-inch monitor, resolution 1024x768, windows xp
- tt-design, tf-design, sanivision
- ms office, ms outlook, ms internet explorer, digital camera software
- laser printer, digital camera, l.a.s.a.r.
- shared laptops, 2 years old, windows 2000, 1024x768, tt/tf-design, c-soft, myosoft; those laptops are shared amongst the employees if they need it directly with the client
- at home his son has a pretty new pc

product relationship
- occasional user of tt-design
- ~1.300 patients/month treated in all branches
- ~50 patients/month treated by him altogether
- 1 patient/month treated by him with tt-design
- 8-10 tf-sockets/month ordered by him via fax — he does not use tf-design
- 70% of all the tf-sockets are ordered by fax, only 30% by email

attitude toward product
- software is too complex, he thinks he is faster with plaster cast

Artifacts

Scenarios

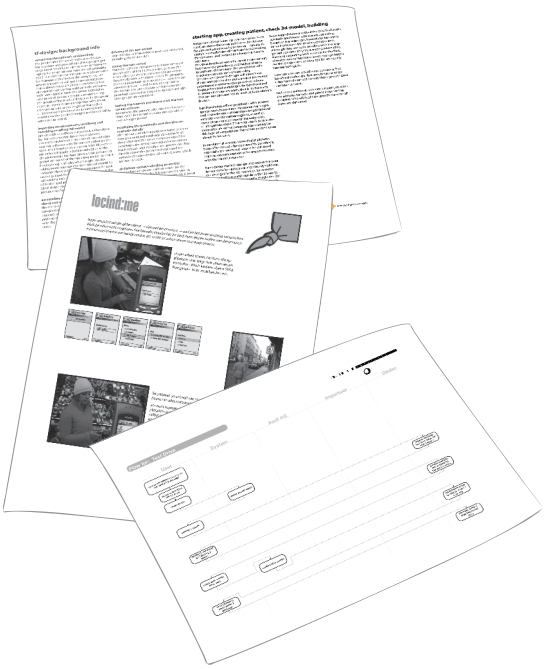
How
Scenarios are descriptions of people and their tasks, they're a mixture of the currently applied procedures and the wishes expressed by the users of how a future system should work. They typically focus on happy day scenarios and don't deal with exceptions. Scenarios are presented in many different ways: narratives, textual stories, storyboards or short videos.
Designers use scenarios to transfer the knowledge gained during user research to the rest of the team.

Why
Scenarios and use cases: use cases can be developed from scenarios. Use cases describe all possible paths. Use case models give an excellent overview of the whole system, but don't transport the priorities of the different cases – this is covered by scenarios.
Scenarios and user stories: user stories are usually much shorter than scenarios and focus on one detailed function. Scenarios provide product managers and engineers with a real world context and a bigger picture of the product.

→ Scenarios (p 80)

Artifacts

Scenarios



Artifacts

Prototypes

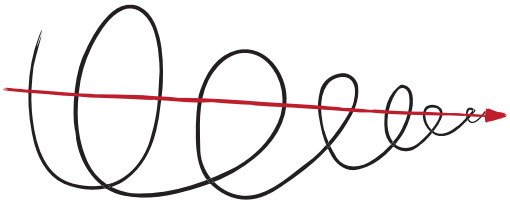
How
Design usually uses throw away prototypes for explorative purposes. Prototypes act as both: a tool for usability testing and an inquiring material to explore possible interactions. Depending on the current project phase and need, the most useful type of prototype should be chosen, ranging from paper prototypes to highly functional ones.
For engineering evolutionary prototypes are more common. Mostly built within the very development environment also the final product is built with. They often act as proof of concept. Discuss the intention of the prototype and exchange the findings of its application to establish a mutual understanding of the different types of prototypes used in the project.

Why
Prototypes are a way of communicating with ourselves, with the users, with the client and with the team members to gather important feedback and findings. Prototypes also often serve as a living guidance for the engineers while implementing the product.

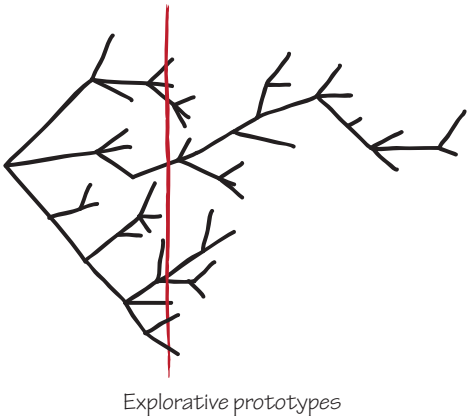
→ Prototypes (p 87)

Artifacts

Prototypes



Evolutionary prototypes



Explorative prototypes

Source: Buxton, 2007

Artifacts

Functional specifications

How
Functional specifications describe the functionality and interaction concepts and are based on the interaction design style guide. Elements are specified in detail in their respective context of use. They define the interaction and its implications for every element and describe what happens in case of an error.
Furthermore they're often the holder for the other specifications, such as flows and pixel accurate screens.

Why
The functional specifications often become the requirement specifications for the engineers. With the distinction that they not only cover the customer requirements, but also the user requirements, flows and pixel accurate screens.

→ Functional specifications (p 106)

Artifacts

Functional specifications

elements of a filter
the FilterHandle can hold more than one filter.

knee function filter **upper assembly function filter**

examples for filter-handles: knee function filter (3 filters each 2 properties), upper assembly function filter (3 filter w/ 2 yes/no properties), adapter material filter (1 filter w/ 3 properties)

filter name
the filter name shows the specification which will be filtered, eg. 'Type of Adapter', 'Material'

properties
the properties consist of the following elements:

- **indicator**: the indicator in front of the CheckBox shows the properties for the selected and rolled over component.
- **CheckBox**
- **property name**

button: detailed filter (only for AssemblyRows)
the material-filter for AssemblyRows has an additional button to choose the material for each component within the assembly. the button is only enabled once an assembly is selected. clicking it opens the AssemblyMaterialDialog.

behaviour

filtering

- properties within a filter are OR-searches
- filters are AND-searches
- all CheckBoxes checked within one filter means the same as no CheckBox checked
- the filter will mark the filtered out components filtered (see ComponentRow) and reorder them so, that the filtered out components are at the very right.
- filtering happens live (checking/unchecking a CheckBox sets the filter)

feedback
on rolling over a component the indicator shows the properties of the rolled over component. if the mouse isn't over an component in the corespondig row the indicator shows the properties of the currently selected component.

switching
to the 2nd filterset can be switched via the SwitchButton in the HeaderStrip. if the 2nd filterset [most likley material] doesn't exist for all rows, no fiter is shown.

disabled row
if the coresponding row is disabled, eg AssemblyRow, the filter isn't shown.

.

Artifacts

Flows

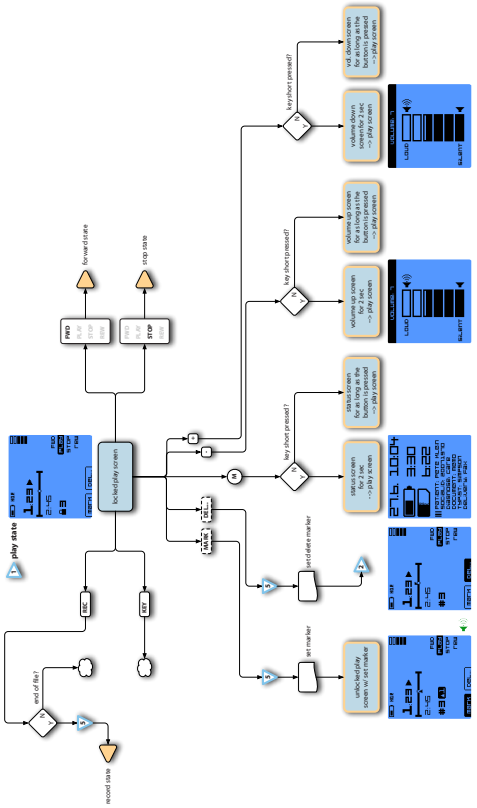
How
Flows cover the details and exceptions, such as error handling, of a product. The syntax used in the flows is defined to the needs of the project and in agreement with the engineers.

Why
Flows are useful for both designers and engineers. For designers they act as a tool for reflection in later stages, when it's time to decide on the details. For engineers they're a part of the specifications.

→ Flows (p 107)

Artifacts

Flows



Artifacts

Pixel accurate screens

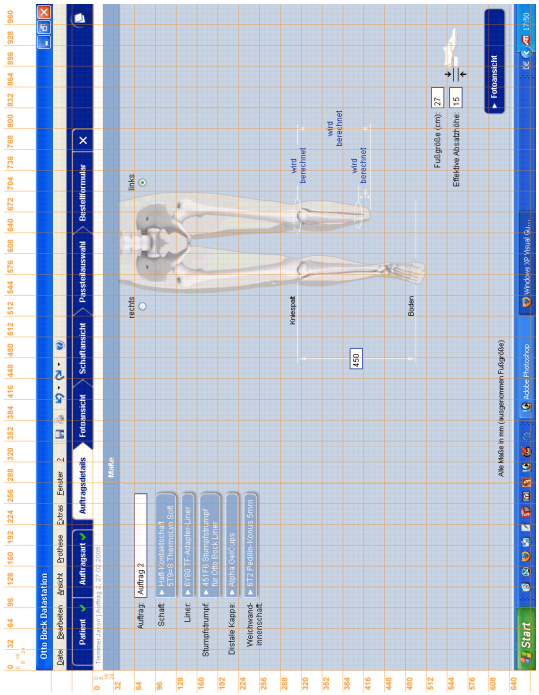
How
Pixel accurate screens represent the static information of the design. It's the responsibility of the designer to ensure that the screens are pixel accurate to remove any ambiguity.
For fixed sized screens/windows a grid overlay with rulers on each side works well. For scalable applications/windows the scalability and arrangement of the elements has to be covered.

Why
Pixel accurate screens remove the ambiguity and are especially important within plan-driven environments. Within agile environments they also should be as accurate as possible, but here the process gives room for a better communication to clarify uncertainty.

→ Pixel accurate screens (p 110)

Artifacts

Pixel accurate screens



Artifacts

Interaction design style guide

How
The interaction design style guide, also known as user interface style guide, specifies guiding principles for the interaction elements used, their application and arrangement, as well as the general look & feel. It's based on the corporate identity style guide of the company and the general interface guidelines the application is running on.
A good user interface style guide can only be built on the basis of a thorough understanding of the users' needs and context of the application and the resulting concepts and interaction design.

Why
The user interface style guide provides guidelines for building new components of the product. Furthermore it acts as training material for new team members – designers and engineers – to get them quickly into the metaphors and drivers of the project.

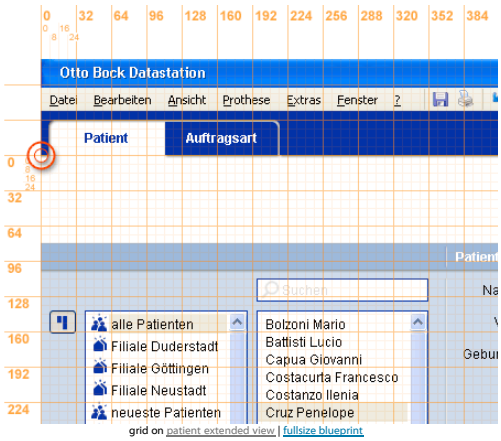
→ Interaction design style guide (p 113)

Artifacts

Interaction design style guide

grid

- main grid: 32x32px
- sub grid: 8x8px
- origin of grid
  - left: screen border
  - top: on the lower edge of the tab row (the origin of the screen is at -87px with standard font size in windows, the height of the tab row is 40px)



grid on patient extended view | fullsize blueprint

rules

- the registration point of elements is always the upper left corner
- try to align elements on the main grid
- align the elements to each other, eg the button 'New Group' in patient extended view is right aligned to the GroupList
- the minimum distance between elements is 8px -- the exception being 4px when we need to be very small
- an element is either a single control or a group of controls -- thus an element in a group of element doesn't need to be on the grid, eg mandatory fields in patient extended view

dialogs

also the elements within a dialog are aligned on the grid. the origin of the grid is the upper left hand corner within the window-border.

Artifacts

Interaction design pattern library

How

An interaction design pattern is a best practice for a recurring design problem. The description of a pattern consists of at least three parts: a problem, its context and a solution – but for better understanding should be accompanied by a rationale and at least one example of use. If possible provide a code example for the implementation of the pattern.
A very complete collection of patterns make up a interaction design pattern language or interaction design pattern library.

Why

The interaction design pattern library provides an overview on many common design problems and may serve as a learning tool for unexperienced team members – and as an inspiration and reference for all team members.

→ Interaction design pattern library (p 114)

Artifacts

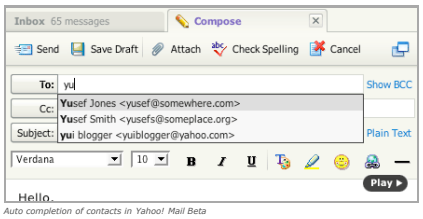
Interaction design pattern library

Auto Complete

Problem Summary

The user needs to enter an item into a text box which could ambiguous or hard to remember and therefore has the potential to be mis-typed.

EXAMPLE:



Auto completion of contacts in Yahoo! Mail Beta

Use When

- The suggestions can be pulled from a manageable set of data.
- The input item can be entered in multiple ways.
- The input item can be matched with a specific data item in the system.
- Speed and accuracy of entry is an important goal.
- The total number of items would be too large or inconvenient for displaying in a standard drop down box.

Solution

Layout

- Use a standard text box for input.
- Label the text box to match the user's expection of what field will be searched against.

Interaction

- As the user types, display a list of suggested items that most closely match what the user has typed. Continue to narrow or broaden the list of suggested items based on the user's input.
- Display the suggested items list in a drop down box directly underneath the text box. The suggested items list may be based on the complete set of data or more narrowly based on other criteria such as each item's frequency of use.
- When available, show multiple fields of information for each suggested item. In the Yahoo! Mail example above, two fields are presented: the contact's full name and the contact's email address.
- Highlight the closest matching item within the suggested items list.
    - Show the matched item as first in the list.
    - Highlight the background of the matched item.
    - When multiple fields are shown for each suggested item the match may occur with the initial characters of any of the fields presented.

Source: http://developeryahoo.com/ypatterns/pattern.php?pattern=autocomplete

Artifacts

Technical documents

How

Provide technical documents – such as technical requirements specifications, software architecture and ER-diagrams – also to team members of other disciplines, especially designers. They provide a basis to better understand the technical constraints and to gain insight into how the engineers see the problem.

Why

It's important to exchange artifacts in both directions, from interaction designers to engineers and vice versa, in order to build up a holistic view of the project.

→ Technical documentation (p 118)

Artifacts

Technical documents

Your Card

How

Why

Your Card

Your Card

How

Why

Your Card