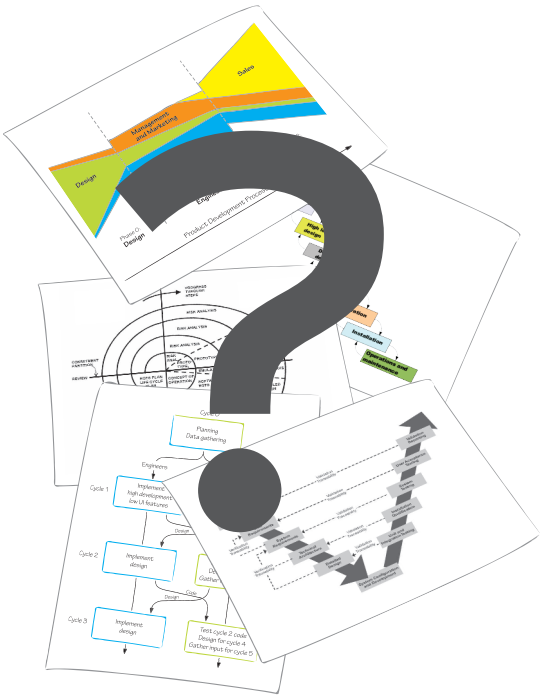
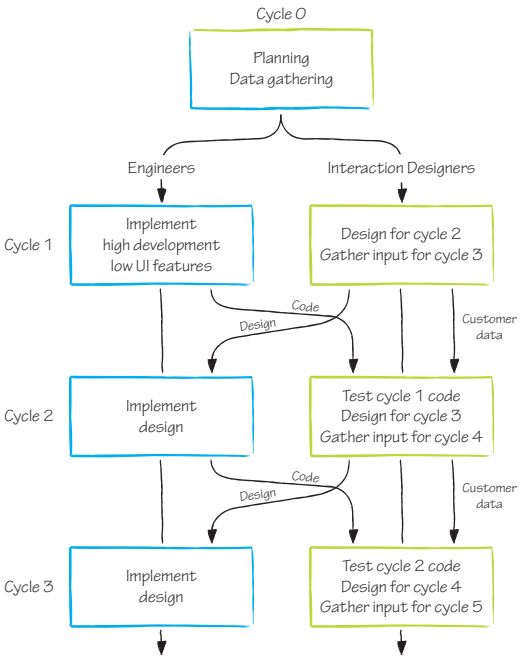
Process

## Question the process



Process

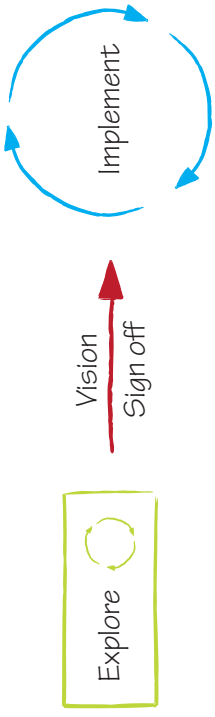
## n±1




Source: Miller, 2005

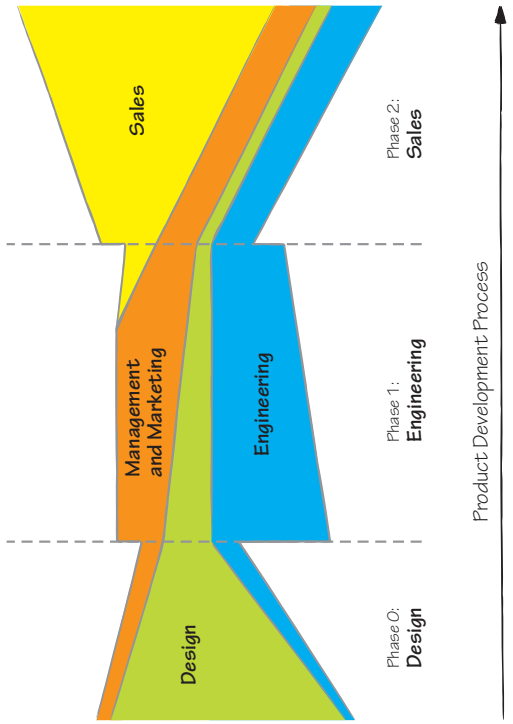
Process

## Combine processes




Process

## Stay in the game




Source: Buxton, 2007

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

## 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

## 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

## 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.

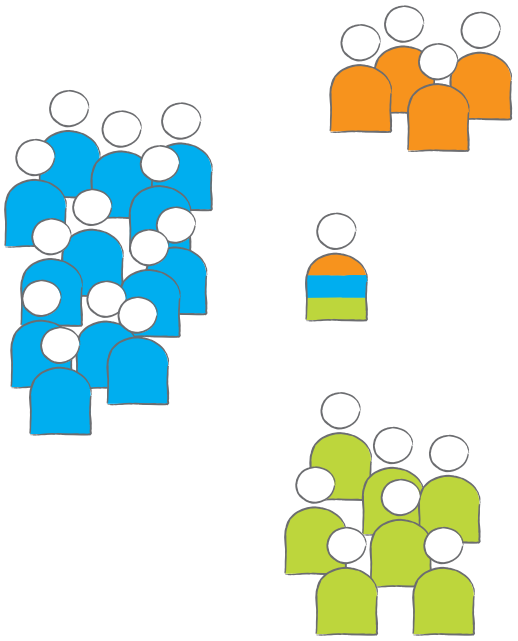
→ Up-front design (p 36)
→ Our experience with processes (p 41)

Process

## Pair design/programming

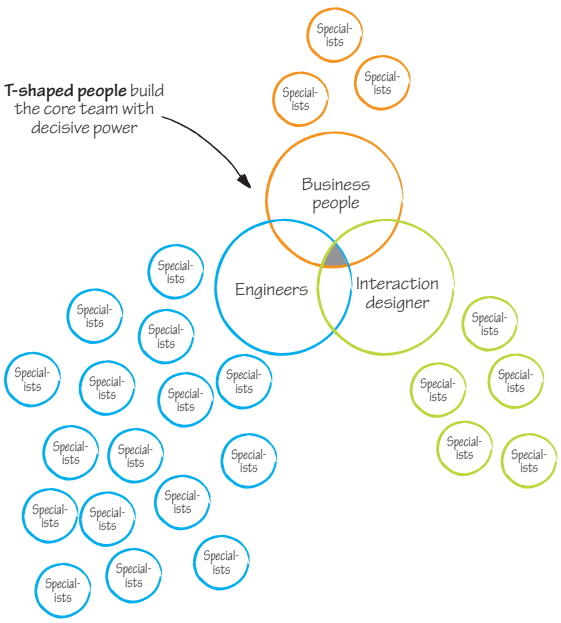
Collaboration

## The catalyst role



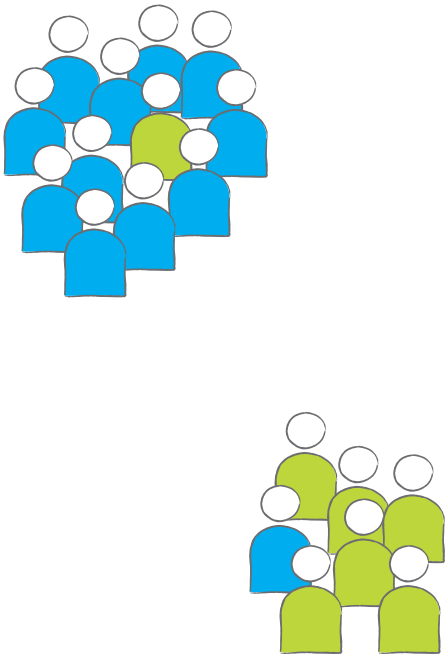
Collaboration

## Power ratio = 1:1:1




Collaboration

## Switch roles



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)

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)

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

## Power ratio = 1:1:1

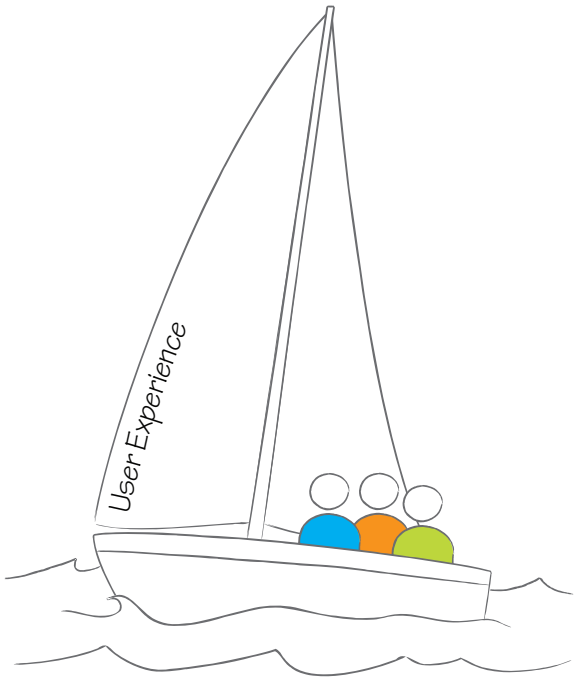
**How**
Balance the power within the team 1:1:1 among the involved perspectives: business to technology to user. The power is democratic within the three perspectives and not the whole team.
The balance can be established by a core team, consisting of a member of each perspective. Ideally the core team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

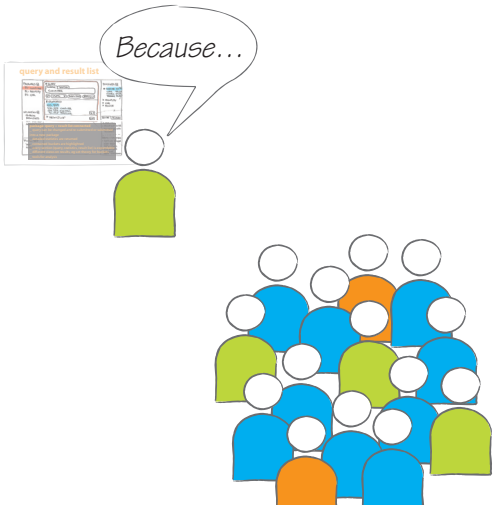
Collaboration

## The user experience is your responsibility



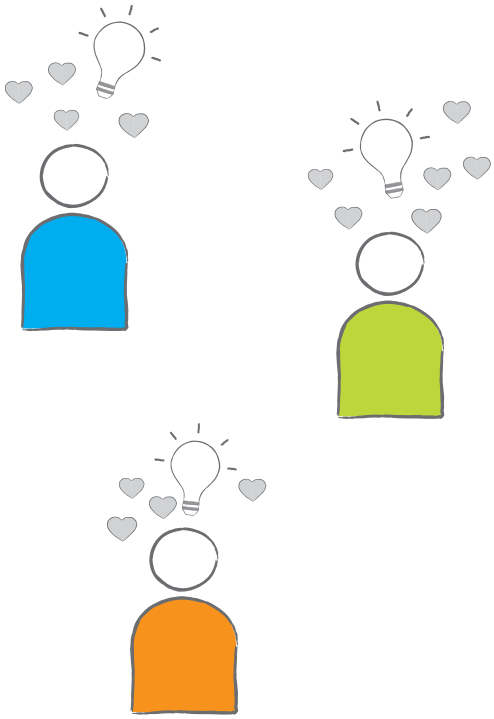

Collaboration

## Give reason



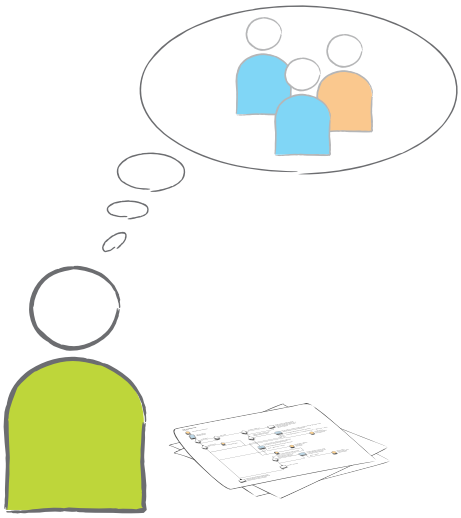

Collaboration

## Kill your darlings – or at least question them



Collaboration

## Design your artifacts for the recipients



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

## 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

## 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

## Kill your darlings – or at least question them

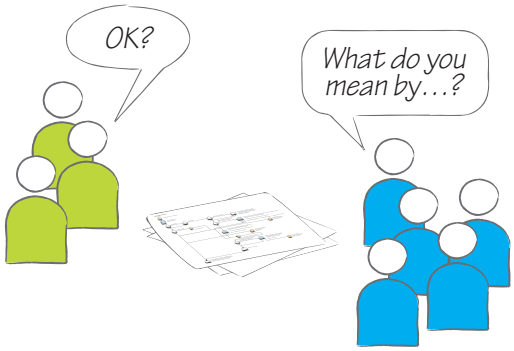
**How**
Do not fall in love with your ideas. Encourage others to question them. Try to get to the bottom of them.
On the other hand: fight for your concepts. Don't throw them over board too quickly, because you get the feedback that they're technically not feasible – question also such a feedback. But 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)

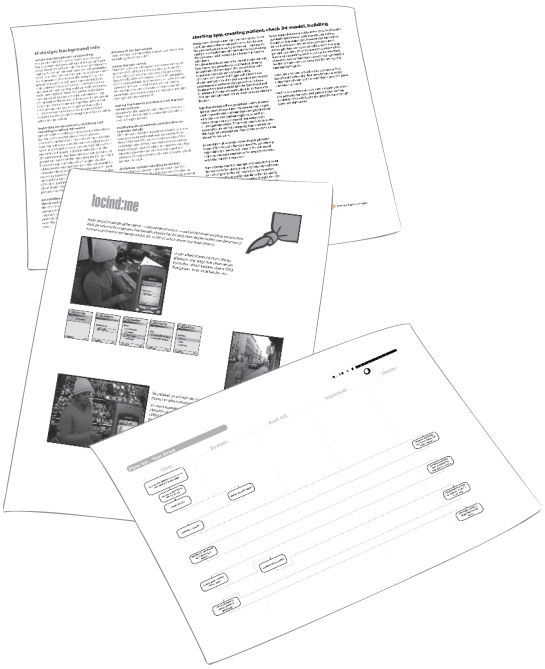
Collaboration

## Specifications – clarify them




Artifacts

## Scenarios




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

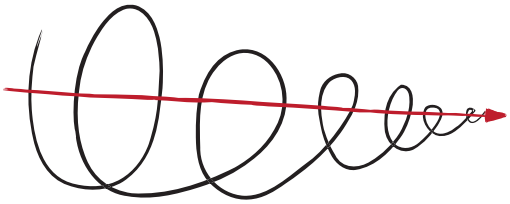
**product relationship**
- occasional user of tt-design
- ~1.300 patients/month treated in all branches
- ~50 patients/month treated by him altogether
- 1 patient/month treated by him with tt-design
- 8-10 tf-sockets/month ordered by him via fax — he does not use tf-design
- 70% of all the tf-sockets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

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

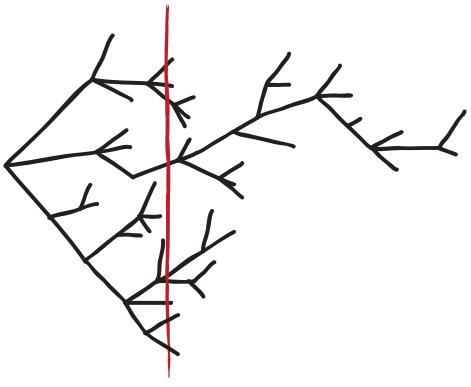
© 2005, GP gregerpauschitz

Artifacts

## Prototypes



Evolutionary prototypes



Explorative prototypes

Source: Buxton, 2007

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)

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)

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

## 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

# 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.

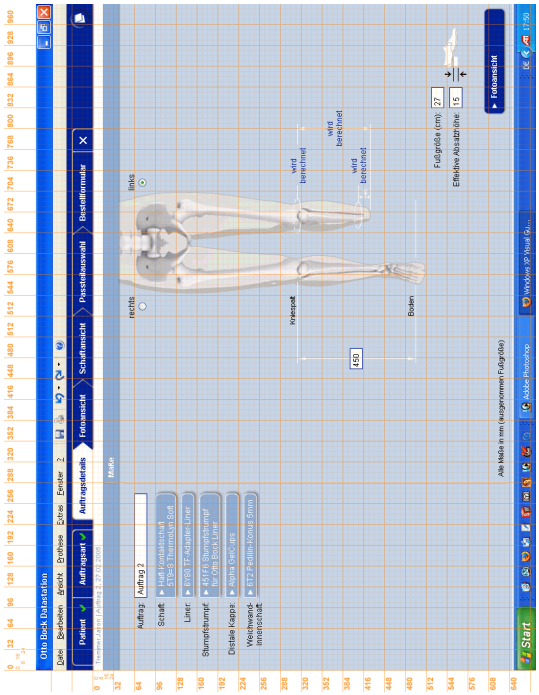
**switching**
to the 2nd filterset can be switched via the SwitchButton in the HeaderStrip. if the 2nd filterset [most likley material] doesn't exist for all rows, no fiter is shown.

**disabled row**
if the coresponding row is disabled, eg AssemblyRow, the filter isn't shown.

-

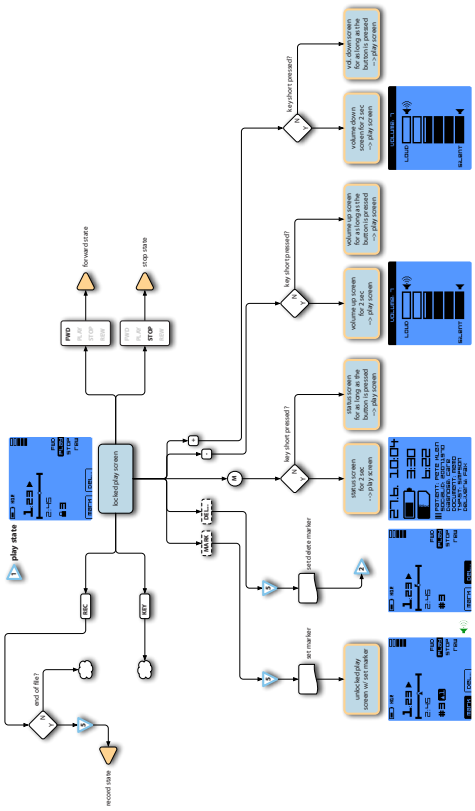
Artifacts

# Pixel accurate screens




Artifacts

# Flows

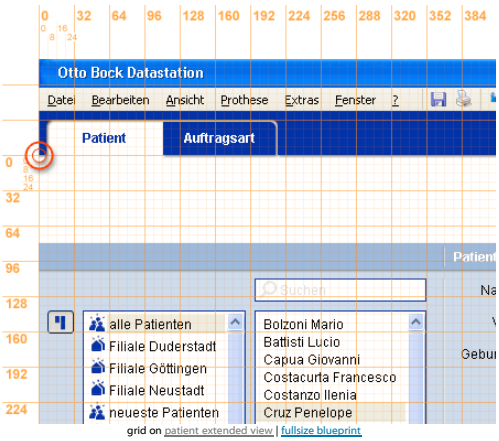


Artifacts

# 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

## 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

## 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 105)

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

## 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

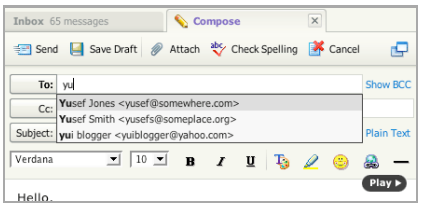
# Interaction design pattern library

## 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://developer.yahoo.com/ypatterns/pattern.php?pattern=autocomplete

Artifacts

# Technical documents

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

## 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)

Your Card

Your Card

Your Card

Your Card

Your Card

How

Why

Your Card

How

Why

Your Card

How

Why

Your Card

How

Why