Structured Approaches to Interaction Design: A Way to Bridge the Gap Between the Results of Foundational User Research and the Final Design of a User Interface

Inaugural Dissertation

submitted to the Department of Psychology of the University of Basel

in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

by

Christian Hübscher

from Basel (BS)

Basel, 2017

University of Basel

Approved by the Department of Psychology
at the request of:

Prof. Dr. Rui Mata (Chair)
Prof. Dr. Klaus Opwis (First Reviewer)
Prof. Dr. Markus Stolze (Second Reviewer)

Basel, 24 April 2017

Prof. Dr. Roselind Lieb

University of Basel

Statement of Authorship

I. I, Christian Hübscher, hereby declare that I have written the submitted doctoral thesis "Structured Approaches to Interaction Design" without any assistance from third parties not indicated.

II. I only used the resources indicated.

III. I marked all the citations.

IV. My cumulative dissertation is based on four manuscripts, of which three manuscripts are already published, and one manuscript is under review.
I certify here that the articles in this dissertation concern original work.
I contributed substantially and independently to all manuscripts in this dissertation.
I have been jointly responsible for the idea, conception, data collection, analyses, and writing of all manuscripts.
This characterization of my contributions is in agreement with my co-authors' views.

Basel, 20 September 2016

Christian Hübscher

Abstract

The present manuscript-based doctoral thesis addresses the question of how approaches to interaction design can be made more structured. This is an attempt to make it more transparent how one can come to a final design of a user interface starting from the results of foundational user research. The results of this analysis can help designers to work very systematic *at times*, to better *reflect* on their own idiosyncratic design process "on the job", *or* to *learn* interaction design in the first place. In *user-centered design* (UCD) there are *lifecycles* and *methods* that already provide a certain degree of structuring with the discrimination of different phases etc. But they leave open some gaps here and there. This work is an attempt to close some of the gaps and bring closer together these individual methods.

Interaction design patterns are a way to describe solutions to problems in designing user interfaces in a very systematic way. However, pattern libraries usually are far from complete. They mostly lack patterns to support the early phases of interaction design and therefore they do usually not link to the abstract models of conceptual design. The *third* manuscript describes an *organizational scheme for interaction design patterns* to support the process of implementing a *complete* pattern language covering all the different levels of the solution domain. The scheme has been found by analyzing several established UCD lifecycles and it has been evaluated by organizing all the individual patterns of several public pattern libraries into it. The *first* manuscript describes a process of systematically building up a pattern language alongside of a redesign project of a complex application in a corporate environment. The *second* manuscript describes how patterns have been evaluated in the aforementioned project, when there were several different solutions for one problem. This is shown with two interaction design patterns for the problem of making required input fields visible to users.

The *fourth* manuscript is an attempt to bring together the idea of a complete pattern language, as a description of the *solution domain* of interaction design, with the different parts of the *problem domain*. Therefore, the same UCD lifecycles (as in the third manuscript) have been analyzed to find a universal structure of the problem domain. Then all the *mappings* between the individual parts of the two domains have been described in order to link the two domains in this direct way. Another way of looking at the gap between the problem domain and the solution domain is by seeing it as a *distance of levels of abstraction* between results of foundational user research and the final user interface. From this point of view a bridging of the gap can be seen in different *intermediate representations* (abstract models, sketches, and prototypes) and linking them together in a coherent way.

These different ways of bridging the gap between foundational user research and the final design of a user interface can be seen as *cognitive artifacts* to foster problem solving and learning of interaction designers.

The original manuscripts are included in the appendix.

Contents

User-Centered Design

User-centered design (UCD) is the part of *human-computer interaction* (HCI) that provides several *methods* and *lifecycles* to support the development of interactive systems. One of the main goals of UCD is a high *user experience* of the solution. The *involvement of end users* into the project is a very important part of this approach. It is advised to do *foundational user research* in order to understand the requirements for a system and to evaluate the designs in *usability tests* with users and refine them iteratively.

User Centered Design Lifecycles

There are several well-established *UCD lifecycles* (e.g. Mayhew, 1999). They are all similar in their setup, even if they have their specialties. The lifecycles describe various *skills*, *methods*, and *work products*. The lifecycles are all structured into distinct *phases* and *iteration* is a very important principle how to move trough them. Usual parts of a UCD lifecycle are: *user research*, *modeling*, *prototyping*, *evaluation*, and *implementation*.

Interaction Design in User Centered Design

The skill of *interaction design*, as part of UCD, is about the translation of the goals and general conditions of a project and the results of foundational user research into a user interface. In early phases of a project a concept for a user interface might be visualized with the help of abstract *models* and later with *prototypes*. The earlier phases of interaction design can also be called *conceptual design* and the later ones *concrete* or *physical design*. Some lifecycles distinguish even more distinct phases of interaction design (e.g.: Garrett, 2002; Baxley, 2002). Another tool to support the task of interaction design (conceptual *and* concrete) are *interaction design patterns*.

Interaction Design Patterns

Patterns are a concept that originally has been introduced by Alexander et al. (1979) to describe solutions to recurring challenges in architecture. Later, Gamma et al. (1995) have applied it to software engineering and finally also interaction designer have adopted the idea (Borchers, 2001). The individual patterns are documented in a *structured format* and usually contain the description of a problem, contextual application rules, a description of the solution and links to related patterns. A *pattern language* is a coherent collection of a limited number of patterns, which can produce a multitude of sound solutions in a certain domain.

Interaction design patterns state solutions in terms of perceived interaction behavior of an interface (Dearden and Finlay, 2006). The individual interaction design patterns are on different levels of a user interface (see Figure 1 for examples). Thus with a *complete* pattern language the *whole* user interface can be "assembled" with a certain number of patterns (see: Alexander, 1979; van Welie and van der Veer, 2003). There are several public interaction design *pattern libraries* (e.g. Tidwell, 2006; van Duyne et al., 2007; Yahoo! Inc., 2009; van Welie, 2009). Most of them are not complete pattern languages. In most cases they mainly focus on the level of *behavior* of the user interface.

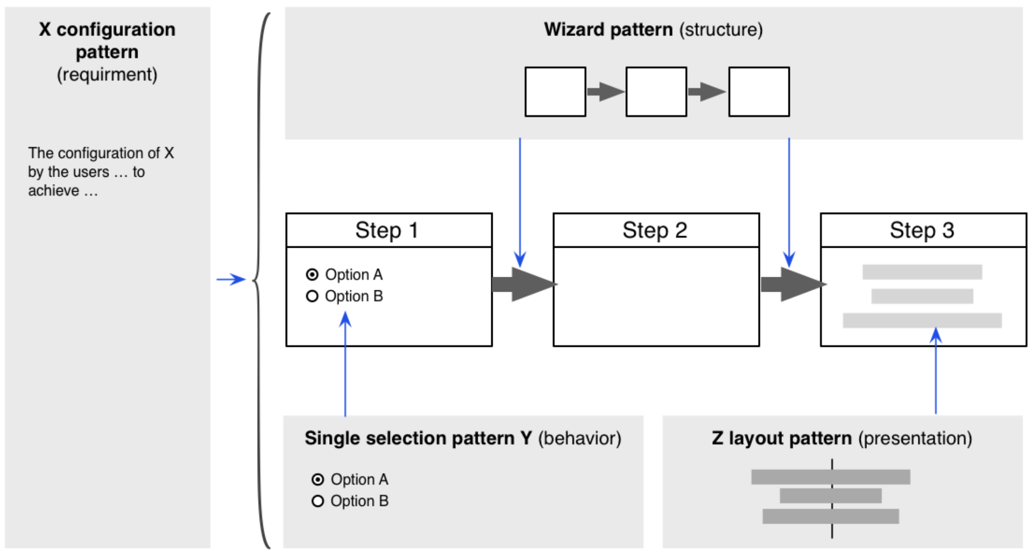

X configuration pattern (requirment)
The configuration of X by the users … to achieve …
Wizard pattern (structure)
Step 1
Option A
Option B
Step 2
Step 3
Single selection pattern Y (behavior)
Option A
Option B
Z layout pattern (presentation)

Figure 1. Interaction design patterns on different levels (from manuscript 4)

Abstract Models for Interaction Design

In the *conceptual design* phase interaction designers often use *abstract models* to visualize certain aspects of a user interface. These models provide a means to visualize *specific* aspects of a user interface in a compact form. A *navigation map* is one example of such a model to visualize the *structure* of a user interface (see Figure 2). There are UCD approaches that provide a rather structured way of linking together some of the different abstract models (e.g.: Constantine & Lockwood, 1999; Roberts et al., 1998). However, they do not the same with prototyping and interaction design patterns respectively.

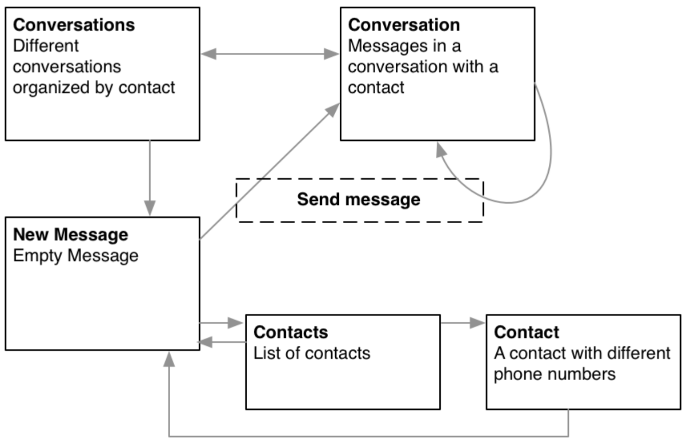

Conversations
Different conversations organized by contact
Conversation
Messages in a conversation with a contact
Send message
New Message
Empty Message
Contacts
List of contacts
Contact
A contact with different phone numbers

Figure 2. Example of a navigation map

Prototypes

In the phase of *concrete design* interaction designers build *prototypes* in order to visualize the user interface in a more tangible way. They can be simple *paper-pencil prototypes* or more sophisticated interactive ones. There are various software tools to support the task of prototyping (see Figure 3 for an example of a prototype). These tools do often contain *some* interaction design patterns, mostly in the form of “atomic” user interface elements and simple modules – mostly on the user interface level of *behavior*. However, in most cases they do not support patterns on other levels.

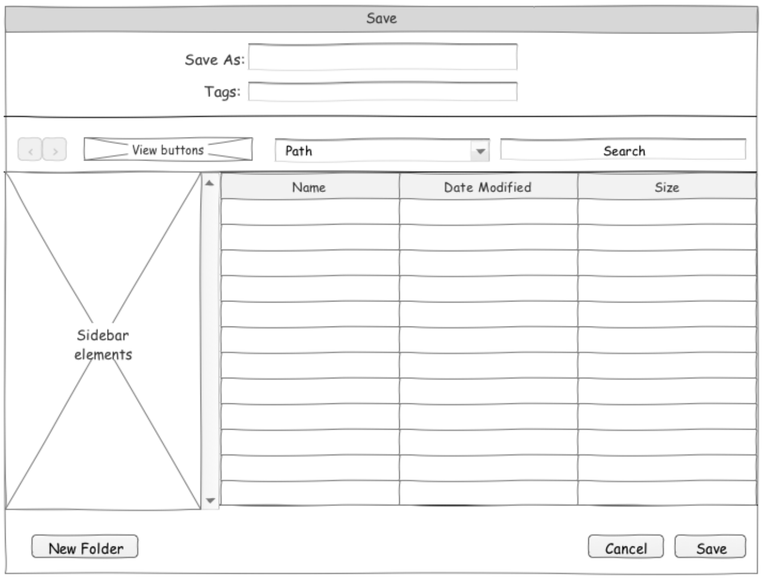


Figure 3. Prototype designed with the tool Axure RP

Structured approaches to interaction design

It could be shown that there are already some parts of interaction design that can be described in a rather structured way. But there is this idea that the *core part* of interaction design is just "too messy" to be understood in an analytical way and that the only way to learn it is by practice. This view might be prevalent in different other design disciplines as well. In the area of architecture, Alexander (1964) also stressed the need for a more systematic approach and confirmed the existence of skepticism among designers against analytic approaches.

Improved interaction design patterns

Interaction design patterns are a means to synthesize a user interface in a systematic way. The descriptions of the individual patterns already are very structured. With a complete and well-organized pattern language all parts of the entire user interface could be described. Unfortunately, in most cases, do designers not have a complete language at hand. Today's pattern libraries especially lack patterns for conceptual design (as will be shown in the third manuscript). So there is a certain gap between the patterns and the abstract models of conceptual design. Having a complete pattern language and working with it in a systematic way would be a big step in the direction of a more structured approach to interaction design. The first three manuscripts will show how this could be achieved.

Gap between problem domain and solution domain

Even if a user interface is fully designed with interaction design patterns, there still is a *gap* between the results of foundational user research and the concept for this user interface. Wood (1997) has described the nature of this gap and has provided solutions to bridge it. The fourth manuscript describes such an approach of combining this bridging of the gap with a systematic way to use interaction design patterns.

About this Doctoral Thesis

The studies for the manuscripts 1-3 have been conducted in the course of a user-centered re-design project of a large corporate client-advisor workbench application at Zürcher Kantonal-bank.

All the manuscripts contained in this thesis are:

1. Pauwels, S. L., Hübscher, C., Bargas-Avila, J. A., & Opwis, K. (2010). Building an interaction design pattern language: A case study. *Computers in Human Behavior, 26*(3), 452-463.

   This work explores how to build and validate a pattern language as part of a user-centered design process in a corporate environment.

2. Pauwels, S. L., Hübscher, C., Leuthold, S., Bargas-Avila, J. A., & Opwis, K. (2009). Error prevention in online forms: Use color instead of asterisks to mark required fields. *Interacting with Computers, 21*(4), 257-262.

   This work shows the validation of one individual interaction design pattern, the marking for required fields, as part of the project mentioned in the first manuscript.

3. Hübscher, C., Pauwels, S. L., Roth, S., Bargas-Avila, J. A., & Opwis, K. (2011). The organization of interaction design pattern languages alongside the design process. *Interacting with Computers, 23*(3), 189-201.

   This work is about finding an organizational scheme for interaction design patterns in order to develop more useful pattern languages. It has been part of the project mentioned in the first manuscript.

4. Hübscher, C., Seckler, M., Tuch, A. N., & Klaus Opwis, K. (under review). Analyzing the gap between the results of foundational user research and the final design of a user interface. [submitted]

   This work explores the nature of the gap between problem domain and solution domain of interaction design together with possible ways to bridge this gap.

Manuscript 1: Building an Interaction Design Pattern Language: A Case Study

Challenge

At Zürcher Kantonalbank the client-advisor workbench application has been developed step by step over seven years. Because of this growth, which was rather organic, the usability of the application has suffered with each release. Then, in the year of 2007 the bank decided to redesign the application in order to improve its usability. Since the application has a plethora of screens, the approach had to be very structured, in order to clean up all the existing inconsistencies. The project team mapped out a *user-centered redesign* process with the goal to also build up an *interaction design pattern language* on the way. The pattern language should inform the redesign at hand but should also be a tool for further development of the application.

A process had to be established to create patterns, validate them, and then bring them together into a pattern library. The existing "patterns" on the many screens of the existing application had to be collected in order to find the ones with the best usability. To have a firm basis for the creation of new patterns, user research, and prototyping had to be done. For the similar patterns solving the same problem an evaluation approach was needed to choose the best patterns.

Results

To develop the pattern language, the following steps have been done (see also Figure 4): A complete *screen-by-screen analysis* of the application's previous version had been conducted to collect all existing interaction design solutions and user interface elements (i.e. "patterns"). To identify the user's context and current problems, eighty-seven *structured interviews* with users have been conducted. The interfaces for the most important tasks and for problematic workflows have been iteratively redesigned with *paper-based prototyping* sessions. When there were different competing patterns the possible variants have been tested with an *interactive prototype* in a *usability lab*. An example of such a pattern testing is documented in the second manuscript.

For the *pattern language* the individual patterns had to be brought together. The initial draft of the language consisted of 136 brief stubs of patterns. The aforementioned prototyping sessions and formal usability tests helped to finalize more and more of the patterns. The final version of the pattern language consisted of 93 interaction design patterns. It was substantially smaller than the initial draft, because the draft also contained the divergent versions of patterns based on the previous version of the application.

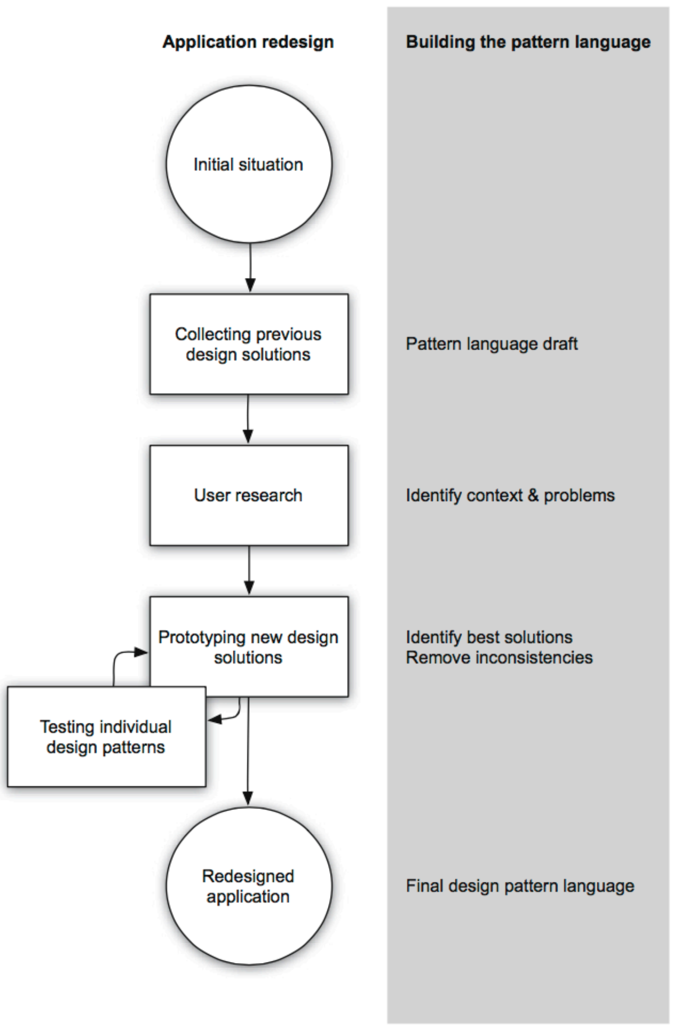

Application redesign
Building the pattern language
Initial situation
Collecting previous design solutions
Pattern language draft
User research
Identify context & problems
Prototyping new design solutions
Identify best solutions
Remove inconsistencies
Testing individual design patterns
Redesigned application
Final design pattern language

Figure 4. The proposed procedure for developing a generative language of design patterns (from this manuscript)

Conclusions

This study shows how the development of an interaction design pattern language can be done in a real-world redesign project in order to do the redesign *itself* but also build a basis for *future* projects. The analysis of the application's previous version for initial versions of interaction design patterns helped to better focus the redesign work itself but also to build up the pattern language.

In order to make the pattern language as *complete*, but also as *compact* as possible, it was important to have a good *organizational scheme* for the patterns. This part of the work will be presented in the third manuscript.

Manuscript 2: Error Prevention in Online Forms: Use Color Instead of Asterisks to Mark Required-Fields

Challenge

In the project described in the first manuscript *possible* variants of some patterns had to be tested with users, in order to find the *best* solution. In the work described here one such case is documented.

In applications, as the one redesigned in this project, there are a lot of forms to fill-in. In most forms there are *required* entry fields and *optional* ones. It is a best practice to mark the required fields as such, in order to help users decide what fields they *have* to fill-in at the least. The analysis of the existing patterns has shown that in the application there were two different ways to indicate required fields:

1. Marking the fields with red asterisks (the de-facto web-standard)
2. Coloring the required fields' background with yellow

The yellow background filled the complete entry field and was consequently a much larger indicator than the small asterisk. Therefore, our hypothesis was, that yellow field-backgrounds lead to faster, more effective, and satisfying form fill-in.

Results

For this study, participants filled out two different versions of a form from the application in an interactive prototype. The study used a *related samples design*. The *independent* variable was the *type of marking* applied to required fields. *Dependent* variables were the number of *errors* a participant made during the task, task completion *time* and a post-test questionnaire to assess a participant's *satisfaction* with the user interface.

It could be shown that yellow-marked backgrounds of required entry fields lead to more reliable and faster fill-in of forms. Furthermore, users reported greater satisfaction with systems if required fields are marked with a colored background. All our hypotheses were supported by these results.

For large and complex input forms, such as some that were implemented in the application to be redesigned, the salient required field markings are an easy and effective way to raise user efficiency and prevent errors. Future research might explore limitations of this finding. In small web forms like simple registration pages, usability might improve less than it does for large transaction forms. Furthermore, the amount of required fields in proportion to non-required fields can play a role in choosing the ideal marking for required fields.

Conclusions

In this project other conflicting patterns have been tested in the same way. So there were these results to back decisions concerning patterns for the most important aspects or with the most difficult-to-solve solutions in order to build a high quality pattern language.

Manuscript 3: The Organization of Interaction Design Pattern Languages Alongside the Design Process

Challenge

If one wants to build an interaction design pattern language for the redesign of a very complex application and to optimally support project staff, which does not have a deep background in interaction design, the language has to have the following properties:

1. It has to be *complete*.
2. It has to be as *generative* but also as *compact* as possible.
3. It should *support a proven approach to interaction design* in a broader sense.

To achieve these three aspects, a *solid organizational scheme* is needed. In analyzing different existing UCD approaches one could find out what is needed:

1. What is the *topology of the solution domain* of interaction design?
2. What are intelligent ways to *chunk* individual patterns? For this it is helpful to analyze existing pattern collections.
3. How can the different types of patterns be *aligned* with proven ways to design user interfaces?

To achieve this, the most prevalent *UCD approaches* (Nielsen, 1993; Rantzer, 1996; Beyer and Holtzblatt, 1998; Roberts et al., 1998; Mayhew, 1999; Constantine and Lockwood, 1999; Garrett, 2002; Baxley, 2002; Cooper et al., 2007) and *interaction design pattern collections* (Tidwell, 2006; van Duyne et al., 2007; Yahoo! Inc., 2009; van Welie, 2009) have been analyzed accordingly.

Results

The analyzed UCD approaches have in common that they differentiate at least *two stages* of interaction design. The *conceptual design* is about working out task flows, the involved user objects, and their relationships and organization. Based on the conceptual design the *concrete design* can be done to design the actual screens of the user interface. Other approaches distinguish even more of such levels.

The approach from Baxley (2002, 2003) goes the farthest concerning its granularity. It has *nine layers* organized into *three tiers*. So it can serve as a useful categorization scheme for patterns, because of its connection to interaction design and its high granularity. To challenge this, well-known pattern collections have been re-categorized using Baxley's layers. Only one group of patterns could not be fit into the layers. However, they could all be taken together in the new category called *requirements patterns*. These patterns were about certain *functions* to enable a user to achieve certain goals (*functional* requirements) or about *how* a user interface is designed, in a general, overall sense (*non-functional* requirements). This extended model is shown in Figure 5. It has similarities with the approach of Garrett (2002).

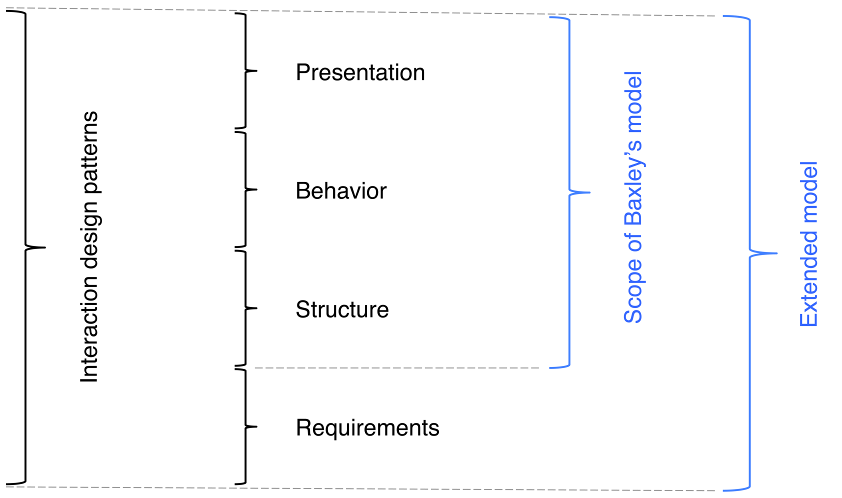


Figure 5. The extended model (from this manuscript)

Conclusions

We have found that this extended model allows a classification of the patterns from the four collections analyzed. Furthermore, the classification helps to advance a pattern language to more *completeness* (see: Alexander, 1979; van Welie and van der Veer, 2003). The classification shows possible *gaps* in the form of unused layers. For example, most pattern collections focus on the *behavioral* tier and provide little support in solving *structural* problems like workflows. This helps to find missing patterns. The fine-grained classification supports the separation of design problems more clearly from one another. Some existing interaction design patterns try to resolve forces from multiple layers, which makes them inflexible and diffuse. To make a pattern language more *generative*, authors should rather decompose such a pattern into individual ones that correspond with different layers.

This organizational scheme helped to further optimize the pattern language in the project described in the first manuscript and to align it with a state of the art approach of designing user interfaces.

Manuscript 4: Analyzing the gap between the results of foundational user research and the final design of a user interface

Challenge

Interaction designers have the challenging task to turn findings of foundational user research into a user interface. In the UCD literature there is a lot of material about how to conduct user research and there is also much advice on generic best practices for the design of user interfaces. In contrary to this, there is not as much found about this *transformation* of the results of foundational user research into a user interface. To capture the nature of this transformation, the *problem domain* and *solution domain* of interaction design have to be analyzed and linked. This fosters the understanding of the *gap* between the two domains and possible ways to bridge it. What is done in the *solution domain* by interaction designers can be described as a choosing from a body of several "building blocks" and combine them together. These blocks can have the form of *interaction design patterns*. Therefore, the organization of a complete pattern language, as described in the third manuscript, can be seen as a description of the *solution domain*. In the work described here, it had to be found a similar description of the *problem domain*. Based on the maps of these two domains, the nature of the gap between the two could be examined.

Results

To show the topology of the *problem domain*, several UCD approaches have been analyzed (the same as for the third manuscript). Based on the description of the two domains it could be shown that the gap between them can be captured in two ways: The gap can be seen as (1) a *distance of levels of abstraction* of the results of foundational user research in relation to the final user interface, or (2) the need of a *mapping* between the individual parts of the two domains. In the existing UCD literature there is a lot of material on how to overcome this *distance* with the help of *different phases*, *iterations*, and *intermediate representations* (models, sketches, and prototypes). Since there is not much material found about a *mapping* between problem and solution domain, this aspect has been analyzed in detail. The analysis of the mapping between the two domains shows how the individual aspects of these two domains are linked. For all individual influences there could be found examples of advice from the literature. See Figure 6 for an example with the mappings on the *structure* level.

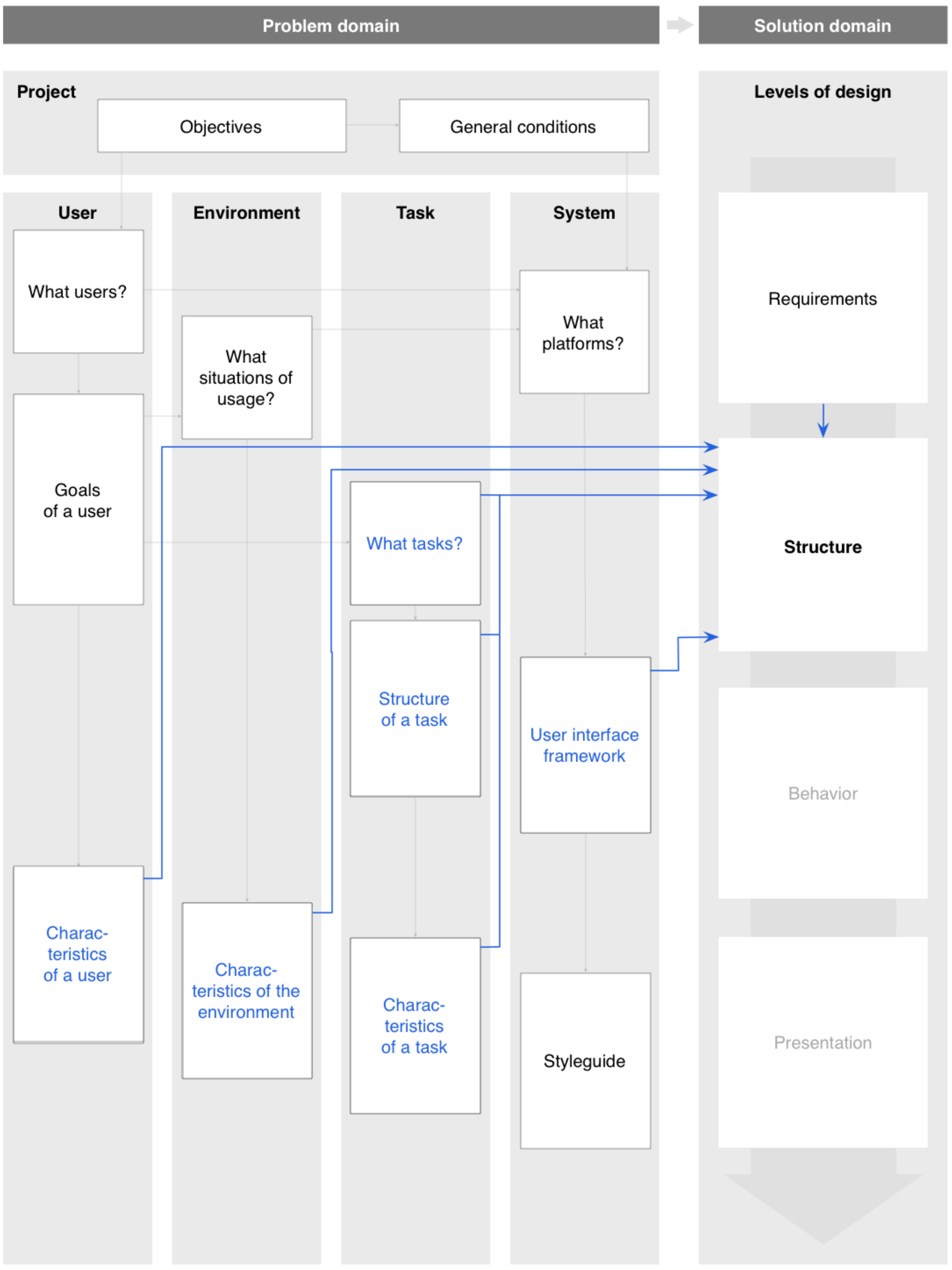


Figure 6. Mapping on the structure level (from this manuscript)

Then it has been explored how this could be brought together with the existing material about intermediate representations. Figure 7 shows how different intermediate representations are spread across the distance of the levels of abstraction between the results of foundational user research and the final design of the user interface. It also shows how they can be mapped from one domain to the other.

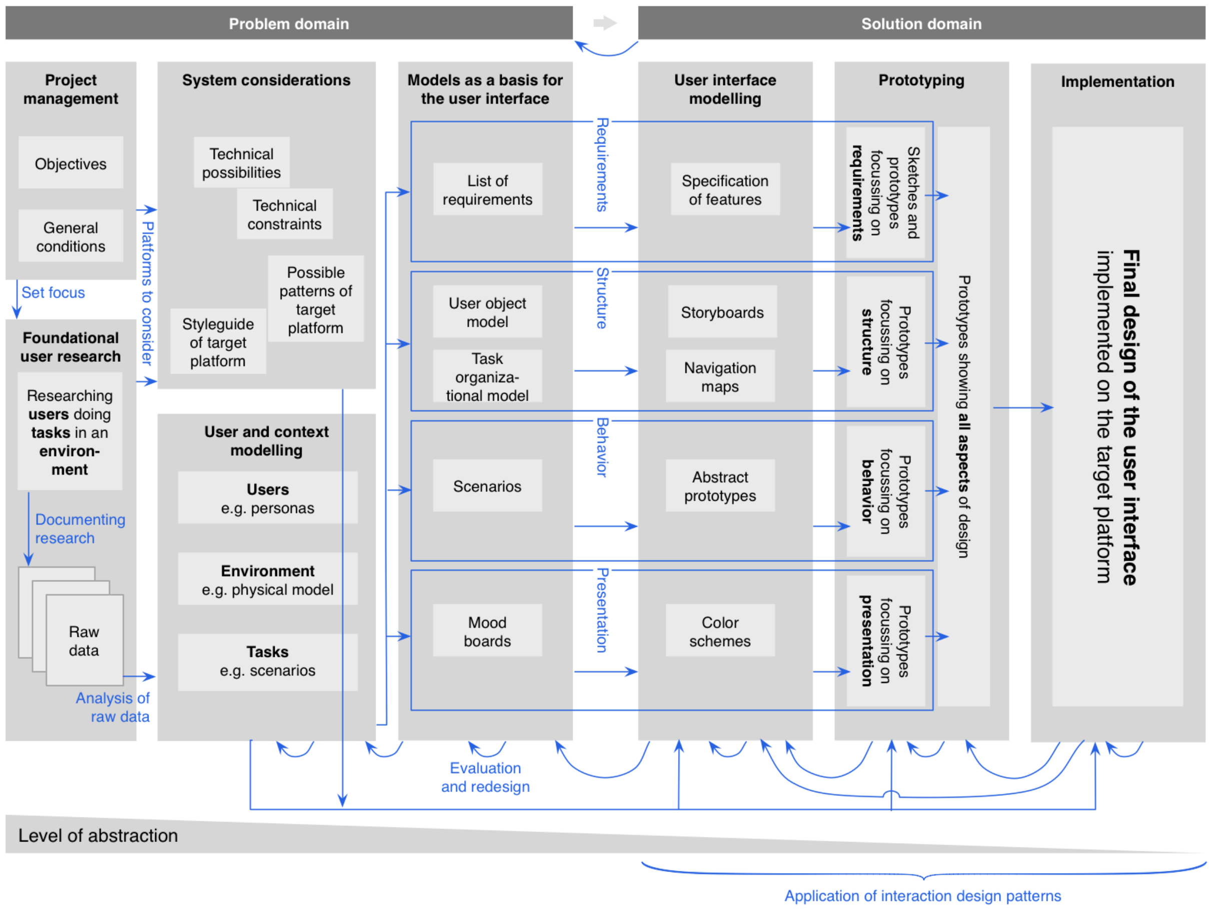

Problem domain
Solution domain
Project management
Objectives
General conditions
Set focus
Foundational user research
Researching users doing tasks in an environment
Documenting research
Raw data
Analysis of raw data
Platforms to consider
System considerations
Technical possibilities
Technical constraints
Possible patterns of target platform
Styleguide of target platform
User and context modelling
Users e.g. personas
Environment e.g. physical model
Tasks e.g. scenarios
Models as a basis for the user interface
List of requirements
User object model
Task organizational model
Scenarios
Mood boards
Requirements
Structure
Behavior
Presentation
User interface modelling
Specification of features
Storyboards
Navigation maps
Abstract prototypes
Color schemes
Prototyping
Sketches and prototypes focussing on requirements
Prototypes focussing on structure
Prototypes focussing on behavior
Prototypes focussing on presentation
Prototypes showing all aspects of design
Implementation
Final design of the user interface implemented on the target platform
Evaluation and redesign
Level of abstraction
Application of interaction design patterns

Figure 7. Combination of mapping and distance to be bridged (from this manuscript)

Conclusions

As the organization of interaction design patterns was a basis for this work, its results can further inform the construction of individual interaction design patterns. Every pattern description should contain the forces, which help to decide whether this particular pattern is a good fit in a certain situation. So these descriptions have to refer to the aspects of the *problem domain* relevant for a certain type of pattern and therefore contain the different mappings.

Even if the gap has been closed to a further degree with the results of this work, it is easy to find several aspects in which the gap can be closed even further.

Final Conclusions

In the first three manuscripts it has been shown how *interaction design patterns* can be used to systematically "assemble" a user interface. The fourth manuscript has shown how the *problem domain* can be mapped out and the user interface can be derived from it in a systematic way. Hence, it can be said that the gap between foundational user research and the final design of a user interface has been bridged to a further degree.

Now the questions arise: Can, one day, a user interface be designed entirely in an algorithmic way? If not, *why* should we then do this work of coming up with even more structured approaches to interaction design? I would argue that in fact it will never be fully possible to design a good user interface in a mechanical way. A free, unstructured process of design will always be needed as an important ingredient in order to really finish a design. Nonetheless, such a detailed understanding of a structured process can be of great value.

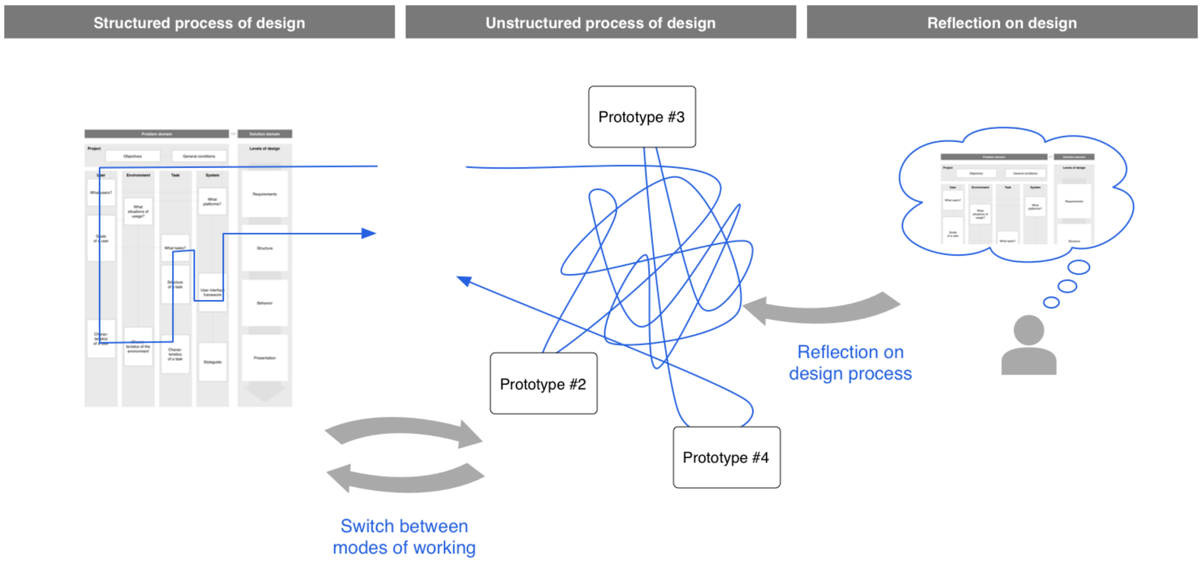

Structured process of design
Unstructured process of design
Reflection on design
Prototype #3
Prototype #2
Prototype #4
Reflection on design process
Switch between modes of working

Figure 8. Two modes of design and the reflection on the design process

I do see two types of benefits we can draw from such highly structured approaches (see also Figure 8): They can (1) map out a *structured process of design* we can follow but they can also (2) provide a means to more effectively *think about* our idiosyncratic process of design in the moment of doing so or to *learn* interaction design in the first place.

Even if the entire structured process cannot be mapped out to every *last* detail, these results still can give some guidance over larger parts of the interaction design journey. The structured process can provide help with the start of a design by gaining a certain *initial overview* over all the relevant aspects, especially in very complex projects. It can also support a certain switching between two “modes of design”. I would say that every designer benefits from switching back and forth between a more *structured* process of design and a more *unstructured* one. Reasons to switch from a structured to a more *unstructured* mode can be to escape *analysis paralysis* or to explore *different variants* of solutions. Switching from a rather unstructured mode into a more *structured* one can be helpful when designers realize *that* a design does not really “work”, but they do not exactly know *why*.

For more effectively *thinking* about the design process the mapping out of the structured approach can serve as a *cognitive artifact* (Norman, 1991). This can be helpful to reflect on ones idiosyncratic design process in the moment of doing design – in the sense of a *reflection-in-action* (see Schön, 1983; Löwgren and Stolterman, 2004). However, it can also be very useful in *HCI education* as a basis to teach interaction design.

References

Alexander, C. (1964). *Notes on the synthesis of form*. Harvard University Press, Cambridge.

Alexander, C. (1979). *The timeless way of building*. Oxford University Press, New York.

Baxley, B. (2002). *Making the web work: Defining effective Web applications*. New Riders.

Baxley, B. (2003). Universal model of a user interface. In *Proceedings of the 2003 Conference on Designing for User Experiences* (pp. 1–14). ACM, New York.

Beyer, H. & Holtzblatt, K. (1998). *Contextual design: Defining customer-centered systems*. Morgan Kaufmann, San Francisco.

Borchers, J. O. (2001). A pattern approach to interaction design. *AI & Society, 15*, 359-376.

Constantine, L. L. & Lockwood, L. A. D. (1999). *Software for use: A practical guide to the models and methods of usage-centered design*. Addison Wesley, Reading.

Cooper, A., Reimann, R., & Cronin, D. (2007). *About face 3: The essentials of interaction design*. Wiley Pub., Indianapolis.

Dearden, A., & Finlay, J. (2006). Pattern Languages in HCI: A critical review. *Human-Computer Interaction, 21*(1), 49–102.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston.

Garrett, J. J. (2002). *The elements of user experience: User-centered design for the web*. New Riders, Indianapolis.

Hübscher, C., Pauwels, S. L., Roth, S., Bargas-Avila, J. A., & Opwis, K. (2011). The organization of interaction design pattern languages alongside the design process. *Interacting with Computers, 23*(3), 189-201.

Hübscher, C., Seckler, M., Tuch, A. N., & Klaus Opwis, K. (under review). Analyzing the gap between the results of foundational user research and the final design of a user interface. [submitted to *Interacting with Computers*]

Löwgren, J. & Stolterman, E. (2004). *Thoughtful interaction design: A design perspective on information technology*. The MIT Press, Cambridge.

Mayhew, D. J. (1999). *The usability engineering lifecycle: A practitioner's handbook for user interface design*. Morgan Kaufmann Publishers, San Francisco.

Nielsen, J. (1993). *Usability engineering*. Academic Press, Boston.

Norman, D. A. (1991). Cognitive artifacts. In J. M. Carroll (Ed.), *Designing interaction: Psychology at the human-computer interface* (pp. 17-38). Cambridge University Press.

Pauwels, S. L., Hübscher, C., Leuthold, S., Bargas-Avila, J. A., & Opwis, K. (2009). Error prevention in online forms: Use color instead of asterisks to mark required fields. *Interacting with Computers, 21*(4), 257-262.

Pauwels, S. L., Hübscher, C., Bargas-Avila, J. A., & Opwis, K. (2010). Building an interaction design pattern language: A case study. *Computers in Human Behavior, 26*(3), 452-463.

Rantzer, M. (1996). The delta method – a way to introduce usability. In D. Wixon & J. Ramey (Eds.), *Field methods casebook for software design* (pp. 91–112). Wiley Computer Pub., New York.

Roberts, D., Berry, D., Isensee, S., & Mullaly, J. (1998). *Designing for the user with OVID: Bridging user interface design and software engineering*. Software engineering series. Macmillan Technical Pub., Indianapolis.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic Books, New York.

Tidwell, J. (2006). *Designing interfaces*. O'Reilly, Beijing.

Van Duyne, D. K., Landay, J. A., & Hong, J. I. (2007). *The design of sites: Patterns for creating winning web sites*. Prentice Hall, Upper Saddle River, 2nd edition.

Van Welie, M. (2009). *Patterns in interaction design*. Retrieved May 7, 2009, from http://www.welie.com/patterns/

Van Welie, M. & van der Veer, G. C. (2003). Pattern languages in interaction design: Structure and organization. In *Proceedings of Interact*, *vol 3*: 1-5.

Wood, L. E. (1997). *User interface design: Bridging the gap from user requirements to design*. CRC Press, Boca Raton.

Yahoo! Inc. (2009). *Design pattern library*. Retrieved May 29, 2009, from http://developer.yahoo.com/ypatterns/

Acknowledgements

I would like to thank the following people for their contributions to my thesis. Without their help and support, this thesis would not have been possible:

- Klaus Opwis, my thesis supervisor, for his support of my work and for bringing HCI to the University of Basel
- Javier Bargas-Avila and Alexandre Tuch for heading the HCI research group and for supporting me with my work
- My co-authors Klaus Opwis, Stefan Pauwels, Javier Bargas-Avila, Mirjam Seckler, Sandra Roth, Stefan Leuthold, and Alexandre Tuch for their valuable contributions to the manuscripts
- Laura Wiles, for helping me with the English proofreading
- My doctoral committee for evaluating this work: Rui Mata (Chairman), Klaus Opwis (First Reviewer), and Markus Stolze (Second Reviewer)
- Most of the research presented in this doctoral thesis was funded by and conducted at Zürcher Kantonalbank – special thanks to Andreas Staub and Roger Huber for making this possible
- The projects in which this research at the Zürcher Kantonalbank was embedded were setup and conducted to a great extent by Stimmt AG
- All the study participants, for their help and patience
- My students at the University of Basel and at the MAS HCID for giving me feedback on my theoretical work and my way of teaching UCD and interaction design
- The many HCI colleagues who gave me feedback on my ideas about interaction design and UCD
- The people who introduced me to the world of HCI back then at the University of Zurich: Thomas Rothenfluh, Daniel Felix and Markus Stolze
- And, finally, my family and all my friends who supported me in various ways

Appendix

1. Pauwels, S. L., Hübscher, C., Bargas-Avila, J. A., & Opwis, K. (2010). Building an interaction design pattern language: A case study. *Computers in Human Behavior, 26*(3), 452-463. https://doi.org/10.1016/j.chb.2009.12.004

2. Pauwels, S. L., Hübscher, C., Leuthold, S., Bargas-Avila, J. A., & Opwis, K. (2009). Error prevention in online forms: Use color instead of asterisks to mark required fields. *Interacting with Computers, 21*(4), 257-262. https://doi.org/10.1016/j.intcom.2009.05.007

3. Hübscher, C., Pauwels, S., Roth, S., Bargas-Avila, J. A., and Opwis, K. (2011). The organization of interaction design pattern languages alongside the design process. *Interacting with Computers 23*(3), 189-201. https://doi.org/10.1016/j.intcom.2011.02.009

4. Hübscher, C., Seckler, M., Tuch, A. N., & Klaus Opwis, K. (under review). Analyzing the gap between the results of foundational user research and the final design of a user interface. [submitted]

Building an interaction design pattern language: A case study

Stefan L. Pauwels [*], Christian Hübscher, Javier A. Bargas-Avila, Klaus Opwis

University of Basel, Faculty of Psychology, Department of Cognitive Psychology and Methodology, 4055 Basel, Switzerland

Abstract

Interaction design patterns are a proven way to communicate good design. However, current pattern collections are not sufficiently powerful and generative to be used as a guide for designing an entire application such as those used in complex business environments. This study shows how we built and validated interaction design patterns that serve as the specification for the redesign of an application. Additionally, they were integrated into a pattern language, as a ruleset for human-computer interaction (HCI) non-professionals to continue development of the application. We demonstrate how individual phases in the redesign of an application can be matched with the process of creating an interaction design pattern language. To facilitate the writing of individual interaction design patterns as well as the development of the pattern language as a whole, a combination of user interviews, controlled experiments and analytical methods has been applied successfully.

Key words: Design patterns, Pattern languages, Interaction design

1 Introduction

1.1 Interaction design patterns

Design patterns describe good solutions to recurring design problems in specific contexts. The concept of design patterns was originally developed by Christopher Alexander as a method of capturing and communicating good

[*] Corresponding author. Tel.: +41-61-2673568.
E-mail address: stefan.pauwels@unibas.ch (Stefan L. Pauwels).

architectural design (Alexander et al., 1979). Each of Alexander's design patterns has a unique name, a numerical ID and gives an overview of the pattern's context and what the solution is about, mostly in the form of a short summary and a picture or figure (see figure 1 for an example). The overview is followed by a detailed description of the problem, how to implement the solution, a rationale why the solution is good and in what context the design pattern should be applied. (Alexander et al., 1977).

Fig. 1. Overview of Christopher Alexander's design pattern *Couple's realm* (Alexander, 2001).

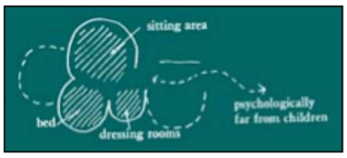
136 . . . COUPLE'S REALM

. . . this pattern helps to complete THE FAMILY (75), HOUSE FOR A SMALL FAMILY (76) and HOUSE FOR A COUPLE (77). It also ties in to a particular position on the INTIMACY GRADIENT (127), and can be used to help generate that gradient, if it doesn't exist already.

The presence of children in a family often destroys the closeness and the special privacy which a man and wife need together.

Therefore:

Make a special part of the house distinct from the common areas and all the children's rooms, where the man and woman of the house can be together in private. Give this place a quick path to the children's rooms, but, at all costs, make it a distinctly separate realm.



The differentiation between problem and context in the detailed pattern description seems noteworthy: Multiple design patterns can solve the same problem for different contexts. Consequently, multiple design patterns can have similar or even identical statements as their problem attribute. Many pattern authors use the term "forces" to describe the constraints that define a problem. By contrast, the context attribute should make clear, when to choose one pattern over the other. This crucial attribute can be defined by a list of conditions that must apply for a pattern to be justified or by a careful description of the context.

The design pattern concept was later adopted by software engineers. Gamma et al. (1995) described a collection of influential software design patterns which are now widely used. Software design patterns differ from Alexander's design patterns in an important aspect: Software design patterns are developed by and for professionals whereas Alexander's architectural design patterns were specifically designed to give non-professionals the power to create good design (Alexander et al., 1979).

In recent years, design patterns have found their way into the field of human-computer interaction (HCI). Early HCI-related patterns appeared at the Pattern Languages of Programming (PLoP) conference and pattern workshops began emerging at the Computer-Human Interaction (CHI) conference (Bayle et al., 1998). Since then many pattern libraries have been published (Tidwell, 2005; Van Duyne et al., 2007; Van Welie, 2008; Yahoo! Inc., 2006) and more are appearing each year. Figure 2 shows an example of a typical interaction design pattern. In the different implementations of the design pattern concept in HCI, the internal structure of a pattern has mostly stayed true to Alexander's pattern form; the attributes' names however vary among implemented design pattern collections. Table 1 shows an overview of typical design pattern attributes used in HCI.

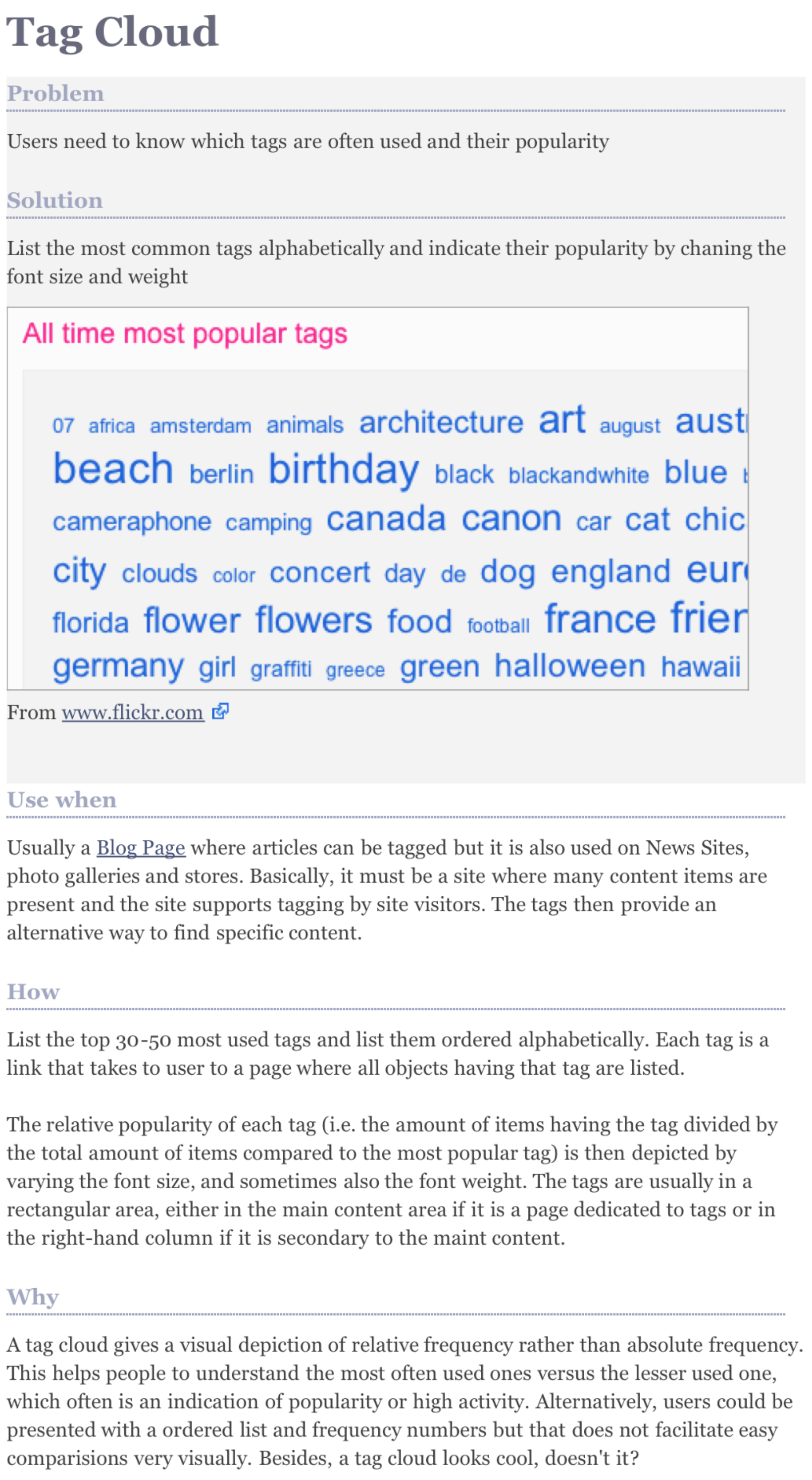
E-learning is another emerging area of application for design patterns that is closely related to HCI. Recent studies analyzed the design of learning environments by evaluating solutions to various problems as design patterns (Van Diggelen and Overdijk, 2009) whereas other studies looked into techniques of finding and writing e-learning and collaborative learning design patterns (Kohls and Uttecht, 2009; Winters and Yishay, 2009).

Dearden and Finlay (2006) proposed the term interaction design pattern to define design patterns in the HCI field because they state solutions in terms of perceived interaction behavior of an interface. This enables a clear distinction between interaction design patterns used in interface design and software design patterns whose solutions focus on source code and software structures.

Similar to Alexander's original design patterns, interaction design patterns are written for professionals and non-professionals alike. Interface design often involves people from a broad, interdisciplinary field of designers, developers, business analysts, researchers and users (Borchers, 2001) who need to have a common understanding of design problems and solutions in order to cooperate effectively. Interaction design patterns enable the communication of design solutions among co-workers of various fields (HCI, IT, business) or users for participatory design (Dearden et al., 2002).

Design patterns are essentially a way of structuring knowledge and not a

Fig. 2. Example of an interaction design pattern (Van Welie, 2008).


All time most popular tags
07 africa amsterdam animals architecture art august aust
beach berlin birthday black blackandwhite blue
cameraphone camping canada canon car cat chic
city clouds color concert day de dog england eur
florida flower flowers food football france frier
germany girl graffiti greece green halloween hawaii

Tag Cloud

Problem

Users need to know which tags are often used and their popularity

Solution

List the most common tags alphabetically and indicate their popularity by chaning the font size and weight

From www.flickr.com

Use when

Usually a Blog Page where articles can be tagged but it is also used on News Sites, photo galleries and stores. Basically, it must be a site where many content items are present and the site supports tagging by site visitors. The tags then provide an alternative way to find specific content.

How

List the top 30-50 most used tags and list them ordered alphabetically. Each tag is a link that takes to user to a page where all objects having that tag are listed.

The relative popularity of each tag (i.e. the amount of items having the tag divided by the total amount of items compared to the most popular tag) is then depicted by varying the font size, and sometimes also the font weight. The tags are usually in a rectangular area, either in the main content area if it is a page dedicated to tags or in the right-hand column if it is secondary to the maint content.

Why

A tag cloud gives a visual depiction of relative frequency rather than absolute frequency. This helps people to understand the most often used ones versus the lesser used one, which often is an indication of popularity or high activity. Alternatively, users could be presented with a ordered list and frequency numbers but that does not facilitate easy comparisions very visually. Besides, a tag cloud looks cool, doesn't it?

method to find new solutions to problems. Solutions described in design patterns need not be new or original but should be proven to work in practice. Consequently, design patterns are not derived from theory but identified as

Table 1
Typical attributes of design patterns.

Design pattern attribute names	Description
Name, Title	Gives the pattern a unique and meaningful name hinting at the solution.
Overview	Makes it immediately obvious to the reader, what the solution is about. If possible this should contain images, such as screenshots or illustrations.
Problem, Goal, What	Describes the problem that has to be solved or a goal that one wants to achieve with the design.
Context, Forces, Use when	When should the pattern be used? Exact description of the context in which the given solution is applicable. This is often stated as a set of “forces” that influence design solutions in the context.
Solution, How, Resolution	What is the solution to the problem, how can the solution be achieved and how does the resulting interface behave?
Rationale, Why, Principle	Why is the given solution favored over its alternatives? How does it resolve contextual forces? Research that supports the solution.
Related pattern	Is this design pattern part of a higher-level pattern? What other patterns does it contain as parts of its detailed solution? Are there similar design patterns that achieve the same goal in similar contexts?
Examples, Known uses	Where has this pattern already been implemented?

invariant aspects of solutions that emerge as best practices. The identification of these invariants is often referred to as *pattern mining* (Dearden and Finlay, 2006).

Successful use of interaction design patterns is reported for example by Lin and Landay (2008), who have used design patterns as a central part of a prototyping tool. They showed that designers who made more use of the available design pattern language were able to produce better results than those using the patterns less or not at all. Borchers (2001) reports another successful interaction design pattern case: Interaction design patterns were created based on results of a user-centered design (UCD) project and were successfully reused later in similar interface design projects. Apart from using interaction design patterns directly for the design process, Hughes (2006) proposes using them to conserve knowledge gained from usability studies.

1.2 Pattern languages

A single design pattern has a small impact on the design process of a graphical user interface (GUI). To leverage the design pattern concept it is usual to integrate multiple related patterns into a pattern library. Some pattern libraries have been published either as books (Tidwell, 2005; Van Duyne et al., 2007) or online (Van Welie, 2008; Yahoo! Inc., 2006). Public pattern libraries such as the above-cited are collections of interaction design patterns of varying size and scope that the respective authors have observed or applied themselves time and time again.

In order to connect the individual design patterns of a library, an important aspect of a design pattern is its relation to other patterns. Thus, rules for a design pattern's use - its context - can consist of references to other patterns. GUI design solutions can be encapsulated through design patterns that inherit from or contain each other, not unlike classes in object-oriented programming. The connection between design patterns is already aparent in Alexander's patterns (see figure 1). Typical interaction design pattern collections link individual design patterns in a "related patterns" section, where alternative solutions to similar contexts or patterns, which include or complete other patterns, are placed. A design pattern can be related to another design pattern in different ways. Van Welie and Van der Veer (2003) distinguish between three fundamental relations:

(1) *Aggregation*: A design pattern can include others that complete it.
(2) *Specialization*: A design pattern can be derived and specialized from another design pattern.
(3) *Association*: Multiple design patterns can occur in the same context or solve similar problems.

If a pattern library can be built that contains the necessary rules for combining the patterns in a way that allows designing a variety of interfaces, the pattern library forms a language. A formal language allows the building of an infinite number of well-formed formulas using a finite number of symbols and rules. A natural language such as English uses words as symbols and grammatical rules, hence enabling the building of an infinite number of correct English sentences. Alexander argues that a pattern language correspondingly allows us to build an infinite number of different but well-formed buildings using a finite set of architectural design patterns as symbols and the patterns' context and connection definitions as rules (Alexander et al., 1979). To advance a pattern collection to the level of a pattern language, a systematic way of connecting individual patterns is needed, which can be used for the implementation of the design formation rules. Such rules should be able to lead through the GUI design process, thus extending the structure to a generative level. This allows designers to move from problem to problem in a logical way, hence designing a GUI by combining patterns according to their application rules, just as we formulate sentences by combining words according to grammatical and semantical rules in a natural language.

Besides providing formation rules for pattern combination, a pattern language needs to be of adequate size to allow for the combination of a finite set of symbols and rules to form an infinite number of designs, just as natural language allows for the production of sentences. But when is a pattern language complete? Van Welie and Van der Veer (2003) argued that a pattern language is complete when every good design we find can be described using it. Based on the language aspect of design patterns, Alexander on the other hand argued that a pattern language can be morphologically and functionally complete: It is morphologically complete when it can account for a complete design, without any missing parts, and functionally complete when it resolves all the forces in the system (Alexander et al., 1979). Other researchers argued that pattern languages should only include significant *big ideas* and not state obvious solutions, thereby denying that pattern languages need or even benefit from a certain completeness. Such a sparse collection of significant design patterns, however, does not cater for a generative language in Alexander's sense.

A pattern language can constitute a valuable design tool for interface design because it is an interdisciplinary field where cognitive scientists, graphic designers and software developers work together. Communication in these interdisciplinary design teams can become a problem. Erickson (2000), Granlund et al. (2001) and others suggested that pattern languages and interaction design patterns can help communication among the different groups involved in design and indeed Borchers (2001) reports the successful use of patterns by teams in interdisciplinary projects. In large enterprises, where GUIs are usually specified by business analysts who develop business requirements to IT solutions, a pattern language can connect requirements to proven solutions and lead to consistent interaction design even with multiple business analysts working on different parts of an application, because people are able to see whether a design pattern describes the solution to their problem in a specific context.

In addition to any connecting structures of design patterns, pattern languages can be organized into categories according to the scale of the problem that they solve (Van Duyne et al., 2007) or thematically, based on the type of user goal that they address (Tidwell, 2005). A categorization by scale additionally allows for a hierarchical organization of a pattern language, linking high-level design patterns to design patterns that deal with details of the former. In HCI, however, applying a hierarchical structure is not a trivial matter because interaction design is not solely geometrically hierarchical but also has to take sequential and timing aspects into account.

Traditional tools to help designers create user interfaces that meet certain quality criteria have included guidelines, standards and principles in various forms. They mostly focus on consistency rather than usability. Some disadvantages of standards, guidelines and principles are given by Mahemoff and Johnston (1998):

(1) Difficulty of guideline interpretation.
(2) Too simplicistic, and not capable of being used effectively by people who are not human factors experts.
(3) Excessive effort required to find relevant sections.

Design patterns can address these problems. Dearden and Finlay (2006) discusses the differences between guidelines and pattern languages in detail and give an overview of how pattern languages can solve the shortcomings of traditional tools. In summary, interaction design patterns have the following advantages:

(1) They include information about the problem that a given solution actually addresses.
(2) They offer details about contextual constraints of a solution.
(3) They explain rationales for solutions, including empirical findings.
(4) They offer the possibility of organizing single interaction design patterns into pattern languages that lead to a generative process of traversing through design problems in varying degrees of detail.

1.3 Building a domain-specific pattern language

The following case study is embedded in a redesign project of a large in-house customer relationship management (CRM) and advisor workbench application. The application is developed and used in a financial institution and has between 3,000 and 4,000 users. In its previous version, it had been developed step by step during a timeframe of seven years and grew larger after each release. It is expected to grow further as more and more applications are integrated into it. Specification of functionalities is done within the company by different people with various backgrounds. However, neither a user-centered design process nor a prescriptive ruleset was applied for development. As a result, many design solutions turned out to be suboptimal and inconsistent. The redesign of the application followed a user-centered design process to address the usability problems that had arisen. A description of the user-centered design process can be found in ISO (1999) or Mayhew (1999). It is also important to note that the application is a customized version of a standard CRM solution, which means that design choices are sometimes limited unless there is good reason to deviate substantially from this standard.

Because the application is expected to be developed further, another goal (apart from a successful redesign) was to make sure that insights gained during user research and prototyping could be preserved for future design projects and that the application's interaction design became more consistent. To achieve both of these goals, we decided to use the application redesign as an opportunity for the development of an interaction design pattern language.

Today, most interaction design pattern languages in the HCI field are sparse collections of some solutions to a large problem space, e.g. the entire web or even user interfaces in general. Although this might be a source of inspiration for GUI design, the generative power of a language that is needed to ensure good and consistent interface design is mostly missing. A pattern language that would be able to constitute such a generative tool has to allow for the generation and description of a variety of possible solutions for the domain for which it is created, addressing design problems of different scales during different design steps. It was our goal to provide further insights into the development of empirically validated interaction design patterns and prescriptive, generative pattern languages and to make a case for doing so in the course of an application design. For our pattern language to be able to satisfy these needs, a sparse collection of big ideas is insufficient. Instead, we strove for Alexander's definition of morphological and functional completeness. Applied to our situation, these completeness criteria can be defined as follows:

(1) Morphological completeness: The pattern language must be able to describe the redesigned application's user interface completely, i.e. the application consists only of solutions described in the pattern language and every pattern is clearly described and/or its possible components are referenced.
(2) Functional completeness: The pattern leaves no forces unresolved, i.e. all problems and requirements identified during the redesign process must be taken into account and solved by the pattern language.

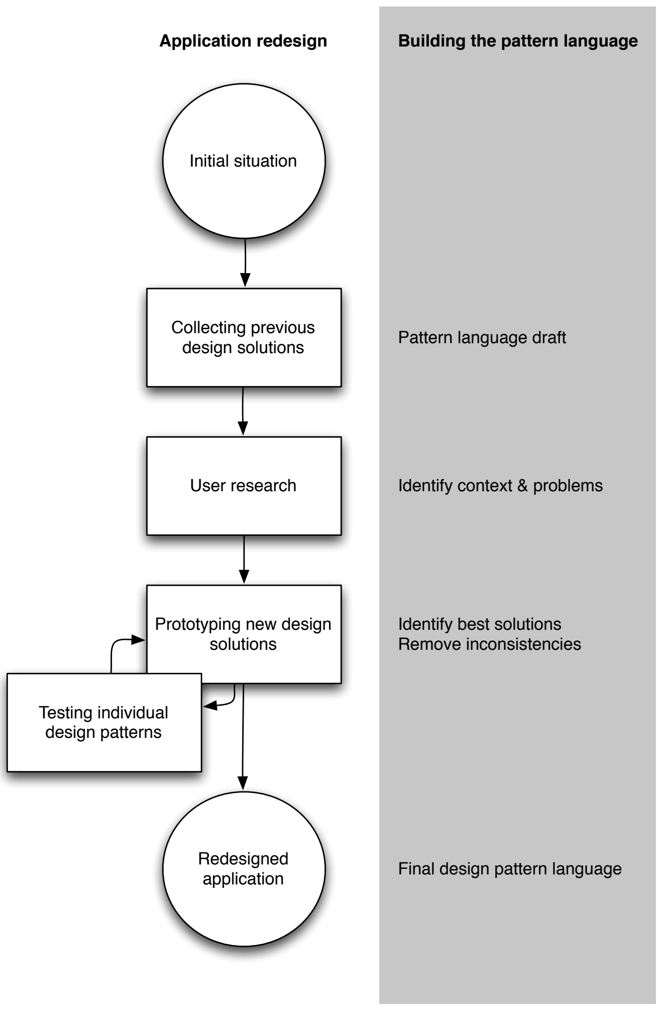
Our hypothesis is that building a complete, validated pattern language for a specific domain takes four basic steps (see figure 3) that addresses all the important pattern language issues:

(1) Collecting previous design solutions (Pattern mining).
(2) User research (Analyzing problems and contexts).
(3) Prototyping new design solutions (Resolving forces).
(4) Testing individual interaction design patterns (Empirical validation of

solutions).

These steps are discussed in detail in the following sections.

Fig. 3. The proposed procedure for developing a generative language of design patterns.


Application redesign
Building the pattern language
Initial situation
Collecting previous design solutions
Pattern language draft
User research
Identify context & problems
Prototyping new design solutions
Identify best solutions
Remove inconsistencies
Testing individual design patterns
Redesigned application
Final design pattern language

2 Collecting previous design solutions

2.1 Method

To achieve a prescriptive interaction design pattern language for the redesigned application, we focused on patterns in the application early on. We wanted to ensure that:

(1) A redesign can be described completely by referencing the pattern language, i.e. describe the latest version of elements once and apply it to the entire prototype and describe screens and task flows that also reference the constitutive patterns.
(2) Future development of functionalities and screens inside the application can continue to use this pattern language, providing a ruleset that preserves the HCI knowledge gained during the redesign.

We collected and categorized all interface elements and design solutions of the application's previous version to obtain initial versions of interaction design patterns. This procedure defined the problem space and provided an initial draft and an estimate on the size of the pattern language.

To get the initial pattern language draft, every single screen of the application was systematically searched for possible interaction design patterns. Because no complete system documentation existed, this was done by hierarchically traversing through the application's structure and collecting business-related task-flow solutions, page types and layouts, visual design, interactions and every element on the page. The collected items were not brought to a pattern structure at this stage, because they were not yet best practice and finding the best solutions to all the design problems was done later. However, we grouped them according to the problem area that they addressed. Using the pattern relating principles of aggregation, specialization and association enabled us to draft patterns referencing essential patterns from other groups. This eliminated redundancy while providing a complete description of solutions on all levels of the interface.

2.2 Results

An overview over the initial draft of interaction design patterns can be seen in table 2.

The initial pattern language draft consisted of design pattern "stubs" with an ID, a name, a short description of the solution and a category indication. They further differed from ideal interaction design patterns in quality: These were not the proven best practices, which design patterns normally constitute, but observed solutions regardless of their usability.

Table 2
Initial pattern language draft: 136 interaction design patterns.

Category	Number of patterns	Category description	Examples
Content patterns	15	Recurring business-relevant task flows	Delegate, Edit
Page type patterns	6	Purpose of a page	Object detail-page
Layout patterns	15	Page-layout elements and areas	Header
Interaction patterns	24	Behavior of GUI elements and GUI element combinations on user input	Filter, single-selection
Visual design patterns	8	Display-related accentuations	Form section divider
GUI elements	68	Basic building blocks of interface elements	Button, Hyperlink
Total	136		

3 User research

3.1 Method

To study the users' most important work tasks and discover where the most severe problems occurred, a user and task analysis (Hackos and Redish, 1998) was conducted. This analysis laid the groundwork for the redesign. Because the studied application's users work in many different sections of the enterprise, finding and understanding a single working context proved difficult and not to the benefit of the different users. To be able to adapt parts of the user interface to the user group that relies upon it most, we chose to apply Cooper's (2004) method of defining and describing user groups as personas. Going into detail of the persona method is, however, not within the scope of this study. We proposed a set of five hypothetical personas with diverging task and problem importance and frequency.

Eighty-seven interviews were conducted during user research. Users from all parts of the enterprise where the application is used were recruited. Every participant took part in an interview before being observed while handling tasks that were both frequent and important for this user's work. In the interviews, participants were asked to specify all tasks that they have to handle and to rate the tasks frequency and importance. During observation of the user handling these tasks, details of problems with the application were collected. All design goals and most of the rationale for design decisions to be made are based on understanding about the different personas' working environments and context, main work tasks and problems that they experienced working with the application. Concerning interaction design patterns, user research should help us to identify not only the problems but also the forces defining a problem's context.

3.2 Results

Analysis of the interviews identified the most important and frequent tasks for all of the personas. Three of the five proposed personas could be confirmed based on analyzed task frequency and importance. The two remaining hypothetical personas showed no differences regarding task importance and frequency. This allowed us to focus optimization of interaction design patterns on specific tasks.

Observing users handling important and frequent tasks showed us the application's main problems, on which we could focus later during the prototyping. As an example, users had difficulty navigating between different sections of the application. The standard software, on which our application is based, left us no choice but to implement the main navigation as horizontally arranged tabs. But because the application has a considerable number of sections (the exact number depends on the user's role), the main navigation needed more space than screen width available. In the previous version of the application, this was solved by making the main navigation as a whole horizontally scrollable in order for the rightmost tab to be reached. Users often had difficulty remembering where a desired section's tab was located and got lost scrolling back and forth in the main navigation.

4 Prototyping

4.1 Method

We started to redesign the application by developing an initial paper prototype. Based on the application's previous version, possible design changes on design pattern level included:

(1) Modifying the solution of an existing interaction design pattern: Although the problem and context remained unchanged, a previous solution with

usability problems was redesigned.

(2) Creating new interaction design patterns: Discovered problems that were not addressed in the application's previous version were solved and added to the design pattern language.

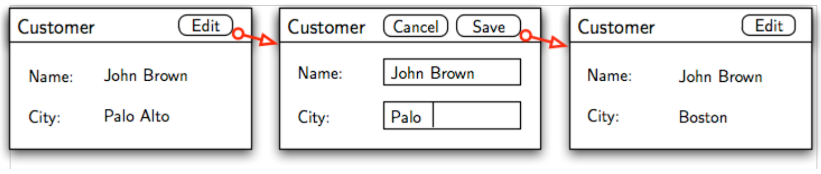
(3) Removing interaction design patterns: Design solutions that were obsolete through redesigning other design patterns were removed from the design pattern language. Also, many solutions for the same problem and context existed, causing the aforementioned inconsistency in the application design. Obsolete and redundant design patterns were removed from the pattern language.

These changes were documented on pattern level. Not every single screen was captured as a design pattern because most of the screens themselves were not solutions to recurring problems, although they could be implementations of a certain page type design pattern. Prototyping took place during five stages. During each stage, 10 users tested a current paper version of the prototype and interaction design patterns. Each stage focused on the user personas' main tasks. The paper prototypes covered all screens that were necessary to complete the tasks. Prototypes (and design patterns) were adapted during stages to react to usability problems that had not yet been solved.

4.2 Results

The resulting prototype consisted of several PowerPoint documents. Slides of the presentation corresponded to steps that the user had to go through to complete the main work tasks. It implemented all the redesigned interaction design patterns.

Fig. 4. The Data Manipulation interaction design pattern.



Data manipulation serves as an example for a modified solution of an existing interaction design pattern: In the previous version of the application, data manipulation was inconsistent. Sometimes, users had to choose to go into edit-mode before fields were editable. In some cases, changes were saved to the database instantly when changing focus of a field, whereas in others the same did not happen until a "Save" or "Submit" button was pressed. The proposed data manipulation design pattern contained an edit-mode and a "Save" Button, to reduce confusion (Figure 4).

5 Testing individual design patterns

5.1 Method

Studies to compare measured usability for different design solutions have been conducted by various researchers. Consider for example Bargas-Avila et al. (2007) who compared different solutions for the presentation of form input error messages. Couper et al. (2004) analyzed differences of usability between various solutions for designing single selection: Drop down boxes, radio buttons and scrollable option boxes. Pauwels et al. (2009) found that required fields in forms can lead to more effective input behavior if the fields' backgrounds are colored.

The common ground of the above-mentioned studies and the pattern testing that we applied in this study is the empirical validation of isolated solutions to interaction design problems. The difference between our pattern testing and other interaction design testing lies (a) in the research goals and (b) the external validity:

(a) In this study, solutions were to be documented as interaction design patterns and became part of a pattern language. Our goal was to close gaps in the pattern language.
(b) Our stimuli were selected with the need to identify the best solution for our application. We had to take several constraints into account which do not apply for most environments. What we identified as a best solution for our design pattern library, therefore, is not necessarily fit for all situations.

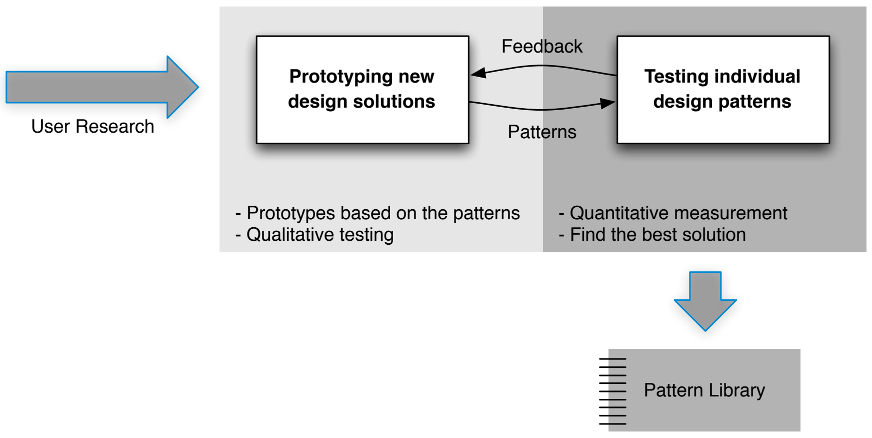
A pattern was included in testing if one of the following cases occured:

(1) The best solution was not clear. Design expertise and prototyping would not identify which of the possible solutions was optimal.
(2) The benefit of a solution was to be shown. Some proposed solutions implied technically complex changes for the IT department. The inclusion of the design pattern in testing was done to explore eventual usability benefits of the solution that justify the costs and implementation effort, especially for deviations from standard solutions of the underlying platform.

On this basis, solutions for seven interaction design problems (see table 3) were analyzed in the usability lab. For each of the seven tested patterns, up to four alternatives were developed. Tasks were selected that contained the patterns in question and prototypes that differed in regard to the tested patterns were implemented.

Because prototyping and pattern testing are two approaches of design validation at different levels of granularity, they were split into alternating phases. This led to ideal feedback of insights from pattern testing to prototyping and of problem statements from prototyping to pattern testing.

Fig. 5. Alternating phases of prototyping and pattern testing.


User Research
Prototyping new design solutions
Feedback
Patterns
Testing individual design patterns
- Prototypes based on the patterns
- Qualitative testing
- Quantitative measurement
- Find the best solution
Pattern Library

Five pattern testing phases were conducted. A minor drawback was that design problems which were introduced at a late pattern testing phase could not be tested with as many participants as early design problems (see table 3), which were also tested in later pattern testing phases to achieve greater statistical power. Eight participants per phase were recruited amongst all users of the CRM application, yielding a total of 40 participants for the whole pattern testing, aged 19-50. Of the participants, 22 were male, with a mean age of 33.5 years (*SD* = 8.5), and 18 were female, with a mean age of 32.2 (*SD* = 8.4).

Table 3
Seven interaction design patterns have been tested empirically

	Number of participants					
Tested pattern	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	Total
Search Input	8	8	8	8	8	40
Deep Links	8	8	0	0	0	16
Multiselection	8	8	8	0	0	24
Required Fields	0	8	8	8	0	24
Wizard	0	0	8	8	8	16
Main Navigation	0	0	0	8	8	16
Filter	0	0	0	0	8	8

The different pattern variants for the selected interaction design problems were tested empirically in a usability lab using clickable prototypes of specific tasks that contained the tested patterns. The prototypes were recreated as part of a mock-up of the CRM application using HTML and Adobe Flash technology. The mock-up was presented on a laptop computer and sessions were recorded with the usability test recording software TechSmith Morae (version 2.0.1). Errors were tracked using markers in Morae. Task completion time was logged automatically by the software. Participants received a paper interview guide that included a short demographic questionnaire, the instructions to the two tasks that they had to complete, and the short version of Questionnaire for User Interface Satisfaction (QUIS) after every task. Finally, two extra questions were added, to explore whether the experimental task was realistic and how frequently the participants encountered it during their work.

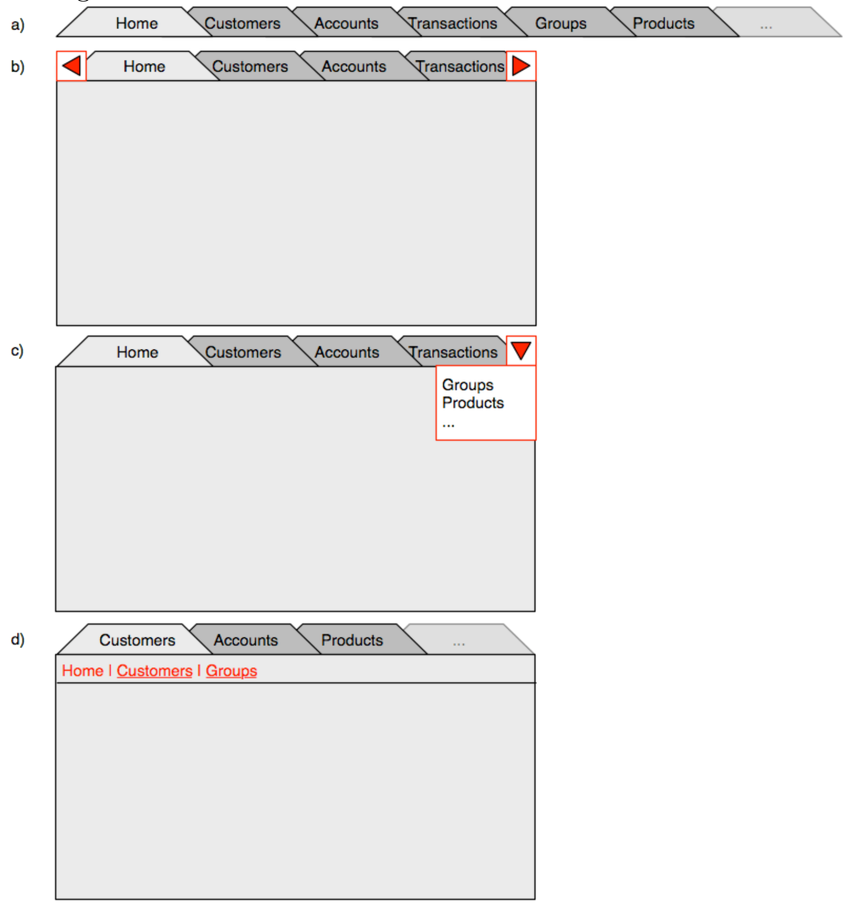
Let us consider the main navigation (figure 6) as an example: It was identified as a problem in user research (see section 3.2). The redesigned application still has a considerable number of sections (figure 6.a). To prevent people from getting lost scrolling through the tabs as in the previous design (figure 6.b), two different solutions for the Main Navigation interaction design pattern were tested: Making extra tabs available through a menu instead of a scrollable navigation (figure 6.c) and grouping the available application sections hierarchically to allow for a two-level navigation (figure 6.d).

Currently three categories of usability measures are common for usability studies: Effectiveness, efficiency and satisfaction (Hornbæk, 2006). Hornbæk recommends using measures of all three groups because there seems to be no implicit correlation between them. We used five usability measures during pattern testing (see table 4).

Table 4
Usability measures applied during pattern testing.

Measure	Category	Explanation
Errors	Effectiveness	Number of errors during task - Wrong button/link clicks - Missing required input - User does not know how to proceed
Task completion time	Efficiency	Time to complete the task
Mouse movement	Efficiency	Distance a user covered with the mouse (in pixels)
Mouse clicks	Efficiency	Number of mouse clicks to complete task
User satisfaction	Satisfaction	The Questionnaire for User Interface Satisfaction (QUIS) is a validated tool for measuring post-use user satisfaction (Chin et al., 1988).

Fig. 6. Different possible solutions for organizing too many tabs were evaluated in pattern testing.



5.2 Results

Interaction design patterns with two alternatives were compared using T-tests. Table 5 shows effect sizes for these tests. Patterns with three or more alternatives were tested using ANOVAs with a factor "pattern alternative". Table 6 shows the percentage of explained variance (partial η^2) of the ANOVAs.

Comparing the different design pattern variants we found medium or large effects mostly for "number of errors". Efficiency was generally less influenced by varying design pattern alternatives. There were also differences in regard to design patterns. Comparing variants for the patterns, Multiselection, Required Field and Main Navigation yielded greater effects than others: Direct Drilldown variants on the other hand did not show differences in any of the measures.

These results suggest that much time and effort can be saved by focusing on user errors while experimentally validating isolated design pattern solutions. Efficiency and, to a lesser degree, satisfaction measures, although commonly used in most system usability tests, are found to be less appropriate for identifying differences in user performance on pattern level.

Table 5
Effect sizes of differences between pattern with two variants.

	Errors	Task completion time	Mouse movement	Mouse clicks	QUIS
Search Input Type	1.76	.04	.21	.03	.07
Required Fields	.59*	.41*	.37	.47	.57*
Main Navigation	1.10**	.28	.36	.42	.3
**:p<.01 *:p<.05					

Table 6
Partial η^2 for ANOVAs of patterns with more than two variants.

	Errors	Task completion time	Mouse movement	Mouse clicks	QUIS
Direct Drilldown	.02	0	.01	0	0
Multiselection	.40**	.12**	.06	.10**	.22**
Wizard	.03	0	.02	.01	.13*
Filter	.54**	.06	.16	.07	.10
**:p<.01 *:p<.05					

For example: Two different new solutions of the main navigation interaction design pattern were tested: (a) Grouping the available application sections hierarchically to allow for a two-level navigation and (b) making extra tabs available through a menu instead of a scrollable navigation.

Table 7
Statistical parameters for Main Navigation design pattern variants.

			Main navigation design
Measures	n	Hierarchical grouping *M* (*SD*)	Overflow menu *M* (*SD*)
Errors	16	1.688 (1.014)	.375 (.619)
Mouse clicks	16	37.250 (8.291)	33.313 (6.019)
Mouse movement (pixels)	16	32,455 (13,835)	28,015 (7,260)
Task completion time (sec)	16	290.594 (74.820)	257.945 (105.864)
Satisfaction (QUIS rating)	16	7.675 (.786)	7.913 (.700)

Table 7 shows observed dependent variables for the two tested solutions of the Main Navigation design pattern. Hierarchical grouping of the application's sections leads to significantly less effective navigation behavior, i.e. users made more navigation errors, $t(15) = 4.392$, $p = .001$ (two-tailed). No significant differences were found for mouse clicks, $t(15) = 1.667$, $p = .116$ (two-tailed), mouse movement, $t(15) = 1.346$, $p = .198$ (two-tailed), task completion time, $t(15) = 1.115$, $p = .282$ (two-tailed), and satisfaction (QUIS) ratings, $t(15) = 1.191$, $p = .252$ (two-tailed) but results nevertheless favored an overflow menu over hierarchical grouping.

6 Final interaction design pattern language

The resulting interaction design pattern language is complete for this application, i.e. it describes every solution and element that may be used during specification of new functionality of the application. This sets our pattern collection apart from published, more sparse collections such as the Yahoo! Design Pattern Library (Yahoo! Inc., 2006) or Designing Interfaces (Tidwell, 2005). Table 8 gives an overview of the final pattern language's scope. In line with our goal to weed out inconsistencies, the redesigned application achieves the same functionality as the previous version using fewer patterns in most of our pattern categories. For example, we reduced the variety of different implementations of business relevant task-flows from 15 to 7 content design patterns.

As an example, figures 7 and 8 show final versions of the interaction design patterns Required Field and Data Manipulation.

7 Conclusions

This study shows that an application redesign offers great opportunities to create interaction design patterns able to form a pattern language that is generative enough to allow for the design of a wide a variety of functionality with a limited number of design solutions. This allows the pattern language to be used as a prescriptive tool for application design. More specifically, a domain-specific and validated interaction design pattern library can be built alongside a redesign process by mining the application's previous version for initial versions of interaction design patterns, analyzing problems and contexts during user and task analysis, resolving the observed forces and validating the results during prototyping.

Through the identification of possible design patterns on the basis of the application's previous version (one of the most time-consuming steps) and collecting

Table 8
Final pattern language: 93 interaction design patterns.

Category	Previous version	Final version	Category description	Examples
Content patterns	15	7	Recurring business-relevant task-flows	Delegate, Edit
Page-type patterns	6	7	Purpose of a page	Object detail-page
Layout patterns	15	8	Page-layout elements and areas	Header
Interaction patterns	24	22	Behavior of GUI elements and GUI element combinations on user input	Filter, single-selection
Visual design patterns	8	7	Display-related accentuations	Form section divider
GUI elements	68	42	Basic building blocks of interface elements	Button, Hyperlink
Total	136	93		

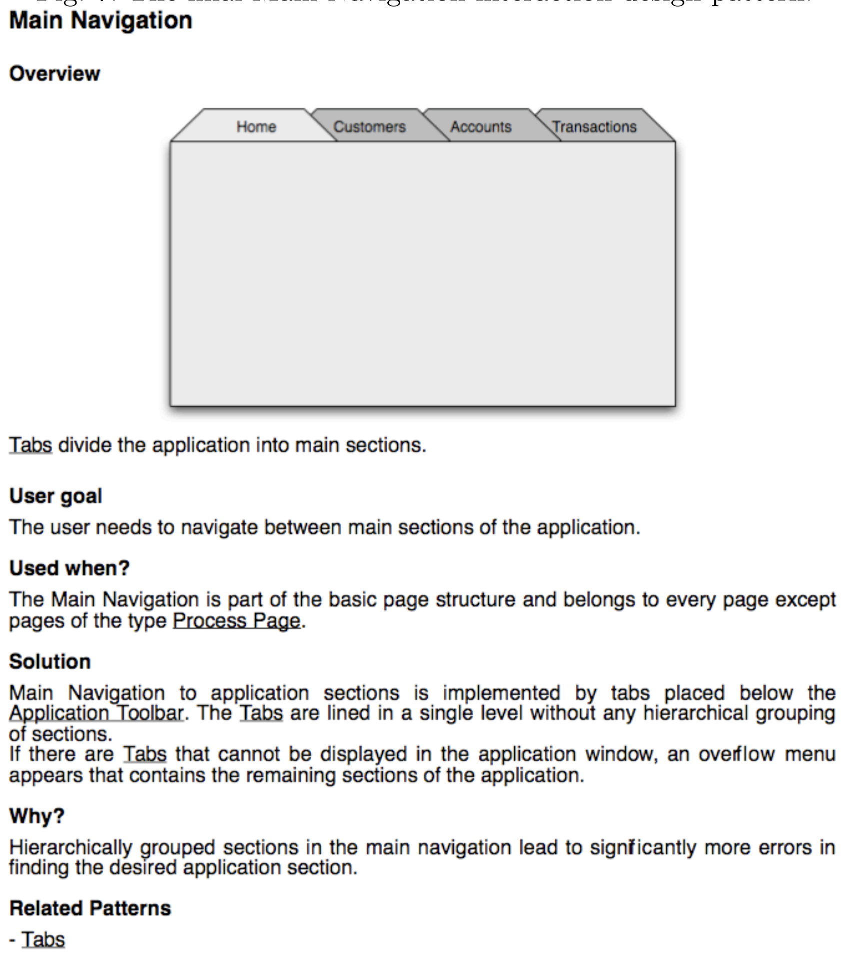
them as an initial pattern language draft, it was possible to estimate the nature and extent of the solutions needed to describe the application. Additionally, it provided insight into basic interactions, task-flows and transactions of the specific business and enabled the design of validated design patterns by analyzing what worked and where the problems were; and subsequently redefining solutions.

We show that completeness of a domain-specific pattern language can be achieved during a user-centered redesign project. Mining an application's previous version systematically for all solutions that it contains, as well as documenting all changes on pattern level guarantees morphological completeness, while conducting user research to identify all problems and verifying changes in prototyping and pattern testing sessions guarantee functional completeness.

However, the aim of the study was to validate the process of building a pattern language, not to produce a domain-independent pattern language which would be valid for various applications. Our pattern language describes the problem space for the continuous development of a specific application. Without this domain-specifity, reaching the level of completeness of our language would not be possible.

The study also demonstrates that to further examine or underpin the pattern

Fig. 7. The final Main Navigation interaction design pattern.


Home
Customers
Accounts
Transactions

Main Navigation

Overview

Tabs divide the application into main sections.

User goal

The user needs to navigate between main sections of the application.

Used when?

The Main Navigation is part of the basic page structure and belongs to every page except pages of the type Process Page.

Solution

Main Navigation to application sections is implemented by tabs placed below the Application Toolbar. The Tabs are lined in a single level without any hierarchical grouping of sections.
If there are Tabs that cannot be displayed in the application window, an overflow menu appears that contains the remaining sections of the application.

Why?

Hierarchically grouped sections in the main navigation lead to significantly more errors in finding the desired application section.

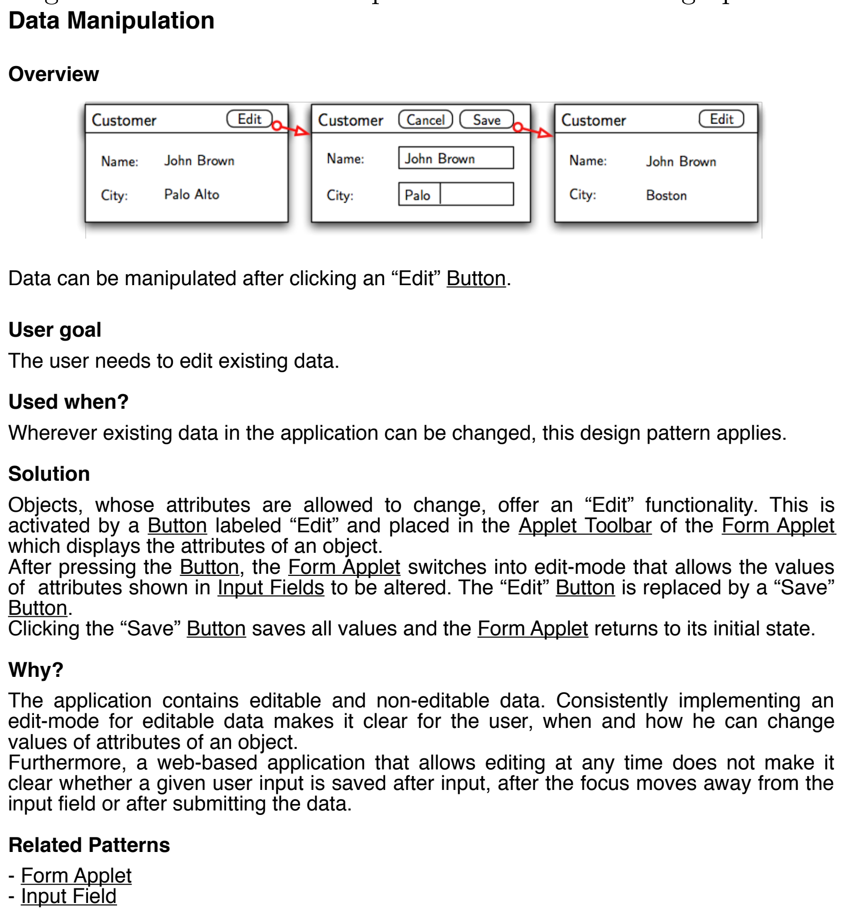
Related Patterns

- Tabs

solutions, experimental pattern testing can be highly effective. This constitutes a way of combining the qualitative, conceptual method of identifying and developing interaction design patterns with quantitative and experimental validation into a complex, integrative approach. The practicability of this approach has been tested and verified by adapting it to a non-trivial but realistic real-world situation, namely the redesign of an aging, inconsistent in-house application, and providing a way of keeping future development of the application consistent and in line with the redesign's intentions.

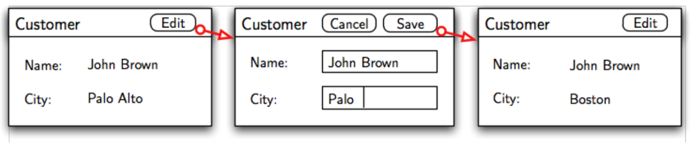
Measuring influences of single design patterns on usability is difficult because alternatives sometimes vary only in minimal ways and a design pattern often accounts for only a small part of the chain of interactions in a complex task. We found that efficiency measures suffer the most because of this and are the least influenced by variations of design solutions. However, user testing of interactions without embedding in a task can easily lead to low validity because it loses relatedness to real situations in which a tested system will be used. Effectiveness and satisfaction measures are less dependent on task selection, because of their focus on outcomes and subjective post-use ratings,

Fig. 8. The final Data Manipulation interaction design pattern.



Data Manipulation

Overview



Data can be manipulated after clicking an “Edit” Button.

User goal

The user needs to edit existing data.

Used when?

Wherever existing data in the application can be changed, this design pattern applies.

Solution

Objects, whose attributes are allowed to change, offer an “Edit” functionality. This is activated by a Button labeled “Edit” and placed in the Applet Toolbar of the Form Applet which displays the attributes of an object.
After pressing the Button, the Form Applet switches into edit-mode that allows the values of attributes shown in Input Fields to be altered. The “Edit” Button is replaced by a “Save” Button.
Clicking the “Save” Button saves all values and the Form Applet returns to its initial state.

Why?

The application contains editable and non-editable data. Consistently implementing an edit-mode for editable data makes it clear for the user, when and how he can change values of attributes of an object.
Furthermore, a web-based application that allows editing at any time does not make it clear whether a given user input is saved after input, after the focus moves away from the input field or after submitting the data.

Related Patterns

- Form Applet
- Input Field

respectively, proved to be of much more value for this type of experiment, with subtle variations in selected elements. This finding is consistent with other comparable studies that analyzed isolated interaction design solutions on a pattern level in a setting where users had to complete realistic tasks, e.g. some of the above-mentioned works comparing solutions on pattern-level (Bargas-Avila et al., 2007; Couper et al., 2004) which found that variation of these solutions had a significant influence on effectiveness and even satisfaction measures but little to none on efficiency.

If there is no possibility of testing designs at an early stage with a complete interactive prototype, testing individual patterns offers a great way of validating solutions to critical parts of a user interface and can complement paper prototyping. We found that the quantitative measuring of these basic building blocks led to valuable performance benchmarks that facilitate design decisions and help justify design solutions to IT departments and management. Based on this research, focusing on effectiveness testing for the experimental validation of single interaction design patterns is recommended.

Beyond building a generative library of interaction design patterns, further research is needed on setting up the library for optimal use. In particular, this study did not try to gain insight into finding the appropriate communication vehicle for such a pattern language. There are three main aspects a design pattern tool can address (Deng et al., 2005):

(1) Pattern catalog: Provide a way to navigate a pattern language, find and validate design solutions.
(2) Pattern management: Manipulate, create and delete design patterns.
(3) Pattern-based design: Provide support for an interface design tool by integrating interaction design patterns as a resource and/or support system.

Identifying an optimal pattern language tool would also imply exploring how software designers would prefer to browse such a pattern catalog structure. Would they look for patterns that address a problem that they encounter? Would they want to drill down a hierarchical tree of design pattern categorizations or prefer to type in keywords to find a solution? This also raises the question, how design by using a complete interaction design pattern language should be integrated into a standard requirements engineering process of an organization with its own software development department.

In future studies, we hope to gain insight into these questions and related issues concerning the practical application of this interaction design pattern library in the studied application's interface design process.

Acknowledgements

The authors would like to thank the Zürcher Kantonalbank (ZKB) in Zürich, Switzerland for the support and funding of this research as part of the UCD ZKBconnect project.

References

Alexander, C., 2001. A pattern language sampler.
URL `http://www.patternlanguage.com/apl/aplsample/aplsample.htm`

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., Angel, S., 1977. A pattern language: Towns, buildings, construction. Oxford University Press, New York.

Alexander, C., et al., 1979. The timeless way of building. Oxford University Press, New York.

Bargas-Avila, J. A., Oberholzer, G., Schmutz, P., de Vito, M., Opwis, K.,

2007. Usable error message presentation in the world wide web: Don't show errors right away. Interacting with Computers 19 (3), 330–341.

Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., Gross, B., Lehder, D., Marmolin, H., Moore, B., Potts, C., Skousen, G., Thomas, J., 1998. Putting it all together: Towards a pattern language for interaction design: A CHI 97 workshop. ACM SIGCHI Bulletin 30 (1), 17–23.

Borchers, J. O., 2001. A pattern approach to interaction design. AI & Society 15, 359–376.

Chin, J. P., Diehl, V. A., Norman, K. L., 1988. Development of an instrument measuring user satisfaction of the human-computer interface. Proceedings of the ACM CHI 88 Human Factors in Computing Systems Conference, 213–218.

Cooper, A., 2004. The Inmates Are Running the Asylum. SAMS, Macmillan Computer Publishing, Indianapolis.

Couper, M. P., Tourangeau, R., Conrad, F. G., Crawford, S. D., 2004. What they see is what we get - response options for web surveys. Social Science Computer Review 22 (1), 111–127.

Dearden, A., Finlay, J., 2006. Pattern Languages in HCI: A critical review. Human-Computer Interaction 21 (1), 49–102.

Dearden, A., Finlay, J., Allgar, E., McManus, B., 2002. Using pattern languages in participatory design. Proceedings of the Participatory Design Conference, 104–102.

Deng, J., Kemp, E., Todd, E., 2005. Managing UI pattern collections. Proceedings of the 6th ACM SIGCHI New Zealand chapter's international conference on Computer-human interaction: making CHI natural, 31–38.

Erickson, T., 2000. Lingua francas for design: Sacred places and pattern languages. Proceedings of the 3rd conference on Designing interactive systems: processes, practices, methods, and techniques, 357–368.

Gamma, E., Helm, R., Johnson, R., Vlissides, J., 1995. Design patterns: Elements of reusable object-oriented software. Addison-Wesley Longman Publishing Co., Inc., Boston.

Granlund, Å., Lafrenière, D., Carr, D. A., 2001. A pattern-supported approach to the user interface design process. Proceedings of HCI International 2001 9th International Conference on Human-Computer Interaction.

Hackos, J., Redish, J., 1998. User analysis and task analysis for interface design. Wiley, New York.

Hornbæk, K., 2006. Current practice in measuring usability: Challenges to usability studies and research. International Journal of Human-Computer Studies 64, 79–102.

Hughes, M., 2 2006. A pattern language approach to usability knowledge management. Journal of Usability Studies 1 (2), 76–90.

ISO, 1999. 13407 human-centred design processes for interactive systems. ISO/IEC 13407.

Kohls, C., Uttecht, J.-G., 2009. Lessons learnt in mining and writing design patterns for educational interactive graphics. Computers in Human Behavior 25 (5), 1040–1055.

Lin, J., Landay, J. A., 2008. Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces. In: Proceedings of ACM CHI 2008 Conference on Human Factors in Computing Systems. pp. 1313–1322.

Mahemoff, M. J., Johnston, L. J., 1998. Principles for a usability-oriented pattern language. Proceedings 1998 Australasian Computer Human InteractionConference. OzCHI'98, 132–139.

Mayhew, D. J., 1999. The usability engineering lifecycle. Morgan Kaufmann, San Francisco.

Pauwels, S., Hübscher, C., Bargas-Avila, J. A., Opwis, K., 2009. Error prevention in online forms: Use color instead of asterisks to mark required fields. Interacting with Computers 21 (4), 257–262.

Tidwell, J., 2005. Designing interfaces: Patterns for effective interaction design. O'Reilly.

Van Diggelen, W., Overdijk, M., 2009. Grounded design: Design patterns as the link between theory and practice. Computers in Human Behavior 25 (5), 1056–1066.

Van Duyne, D., Landay, J., Hong, J., 2007. The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-centered Web Experience. Addison-Wesley Professional.

Van Welie, M., 2008. Patterns in interaction design.
URL `http://welie.com/patterns`

Van Welie, M., Van der Veer, G., 2003. Pattern languages in interaction design: Structure and organization. Proceedings of IFIP INTERACT03: Human-Computer Interaction 3, 1–5.

Winters, N., Yishay, M., 2009. Dealing with abstraction: Case study generalisation as a method for eliciting design patterns. Computers in Human Behavior 25 (5), 1079–1088.

Yahoo! Inc., 2006. Yahoo design pattern library.
URL `http://developer.yahoo.com/ypatterns`

Error prevention in online forms: Use color instead of asterisks to mark required fields

Stefan L. Pauwels [*], Christian Hübscher, Stefan Leuthold, Javier A. Bargas-Avila, Klaus Opwis

Department of Psychology, University of Basel, Switzerland

Abstract

In this study, a simple but important user interface design choice is examined: When marking required fields in online forms, should GUI designers stick with the often used asterisk that many form design guidelines cite as the de-facto web standard, or should they choose a colored background as a new design solution to visually signal which input fields are required? An experiment with 24 participants was conducted to test the hypotheses that efficiency, effectiveness and satisfaction ratings of colored required fields exceed those of asterisk-marked required fields. Results indicate that colored required field marking leads to fewer errors, faster form fill-in in and higher user satisfaction.

Key words: Online Forms, Required Fields, Error Prevention, User Feedback, Interaction Design

1 Introduction

Entering data into forms is the information worker's equivalent to the factory worker's placing items on a conveyor belt. Both computer applications and machines installed along the conveyor belt are fed input, data or items, based on which they carry out certain transformations. In order to avoid process errors (e.g. transformations applied to the wrong item, yielding undesirable or at least unpredictable results), both computer applications as well as factory machines need to stop processing and alert the worker if no input is present. If input is present and properly formatted as required by the transformations applied, both types of machines work with great speed and accuracy.

[*] Corresponding author. Tel.: +41-61-2673568.
E-mail address: stefan.pauwels@unibas.ch (Stefan L. Pauwels).

As Frederick Winslow Taylor, the founder of scientific management, argued, companies should strive for the greatest possible productivity, because this leads to the greatest possible prosperity for company owners and workers alike (Taylor, 1911). Productivity can be defined as data or items processed per time unit (US Federal Bureau of Justice Assistance, 2008). Thus, for computer applications, in order to achieve greatest possible productivity, companies need to prevent errors in data input and design input systems to maximize input and transformation efficiency. Knowing what input a process needs in order to proceed and how exactly to enter data in the correct format are the two main problems form design needs to address from the information worker's perspective. This study examines a part of the first question: How should required input fields for data input on an online-form be marked?

Wroblewski (2008) mentions three different usage contexts of online forms: e-commerce applications, applications for social interactions, and productivity-based applications. He finds that form design should not be neglected by companies with online user interfaces, stating increased completion rates of between 10% and 40% from improvements indicated by his research and experience. Spool (in Wroblewski, 2008) mentions a real-life example of an online-form redesign yielding increased revenues of $1.5 million in the first week, with a total of over $300 million increased revenue in the first year.

1.1 Theoretical Background

Cooper et al. (2007) devote the first chapter of their textbook on user interface design to goal-directed design, which rests on the awareness that human behavior is goal-oriented from the age of three (Klossek et al., 2008). Behaving in a goal-oriented way means that users choose their actions in order to get as much value as possible from an IT system with the least amount of effort expended. As for forms, they will try to enter the least possible amount of data in order to get the desired result, e.g. start or proceed with a process or become authorized to do something. The only reasons to enter more information are either errors in input field validation, or the system's failure to interpret the user's input in such a way as to arrive at a single possible path of the process, therefore asking for more information. Cooper et al. (2007) argue that it is best to provide the user with rich modeless feedback, signaling that required data is missing without interrupting the user's work.

Most web design guidelines are derived from a rather small set of heuristics like "Minimize the user's memory load" or "Provide an obvious way to undo actions" (e.g. Nielsen and Landauer, 1993; Polson and Lewis, 1990). Nielsen (1994) shows that a few heuristics are indeed enough to explain a remarkable number of observed usability errors. Tidwell (2005) presents twelve behavioral patterns gathered from user observations, and derives the heuristic of marking required input fields from the pattern of “deferred choices”, stating that users often want to or have to skip input fields and may want to come back to them later. Considering the five usability metrics introduced by Nielsen (1993), the marking of required fields can be assumed to contribute mainly to efficiency and error prevention, which, for online forms, translates into four steps:

(1) The user’s perceptive and cognitive acts of finding the required fields with respect to the current task
(2) Entering properly formatted data into the input fields as quickly as possible
(3) Navigation between form input elements
(4) Validation of form input, where each blank required input field and each formatting error decreases efficiency substantially

The first step is addressed by all web design guidelines examined for this study, which argue that required fields should be clearly marked in order to make users efficient and prevent errors (Fowler et al., 2004; Horton, 2005; Koyani et al., 2004; Shneiderman and Plaisant, 2004; Tidwell, 2005; Wroblewski, 2008). One exception is found in the Apple Human Interface Guidelines (Apple Inc., 2008), which states that, to prevent visual clutter, an asterisk or custom icon next to a required input field should be displayed only after the user attempts to leave the current context (e.g. by clicking Continue or OK).

Whereas Horton (2005) is satisfied with marking required fields, other guidelines explicitly state design solutions: Wroblewski (2008) argues use of an asterisk, because it has become the de-facto standard on the web, and, for top-aligned input field labels, even the use of “* required” instead of the asterisk alone. He warns not to use the same indicator for required fields as for optional fields throughout a web site. Fowler et al. (2004) also argue for the asterisk, but mention the use of arrows or other symbols to indicate required fields. They advise against using color (for either field background or label) or boldface, because screenreaders are not able to interpret their meaning, necessitating a lot of guesswork for the visually impaired or blind people using screen readers. Shneiderman and Plaisant (2004) state having taken their guidelines from practitioners’ works, since there is “a paucity of empirical work on form fillin”. They argue that optional fields should be marked with the word “optional” or another distinct visual mark, and that optional fields should be placed after required fields. In addition, they recommend using a clear completion signal, so users understand at what point they can safely submit the form because all necessary information has been entered.

In the only empirical study that has been identified to date, Tullis and Pons (1997) compared several possibilities for marking required input fields. They found only small differences in completion time between chevrons placed before the field label and colored fields, but users preferred colored fields to chevrons in ratings on scales of visual appearance and overall effectiveness.

As for the second and third steps, entering data in the right format and navigating between input elements: Because expert users are fastest if they don't have to home between mouse and keyboard, speeding up form input entry is a main priority of form design. Shneiderman and Plaisant (2004) advise enabling the Tab key to move the cursor between fields. Many other guidelines for form entry and form navigation can be found in the works quoted above, like considering the layout of form input elements or using effective default values. For this study, it is sufficient to state that if users are able to enter data as quickly as possible, they are faster the fewer input fields that are required and the fewer optional ones they have to attentively process in order to rule them out and focus on the required ones. The most efficient form is the one that has the fewest possible required fields grouped together, clearly indicating that users can submit the form once they have completed the fill-in of the group. This line of argument is found in several works, e.g.Fowler et al. (2004) and Wroblewski (2008).

Concerning the fourth step, validation of form input, Bargas-Avila et al. (2007) researched the presentation of error messages but not the marking of required fields. Their Modal Theory of Form Completion suggests that users enter data into a form while in completion mode before being mentally ready to address errors in revision mode. It is not known whether required field markings support users during completion mode, revision mode or both. Since Bargas-Avila et al. (2007) show that users tend to just overlook messages relevant for evaluation during completion mode and their form fill-in performance is not negatively influenced, it can be assumed that good required field design cannot do any harm even if unnecessarily present in completion mode.

Practitioners in form design mention the trade-off between efficiency and error-prevention (Baxley, 2002): Forms can either be optimized for expert users, who know data type and format of every single entry field, or they can be optimized for novice or infrequent users in order to prevent data entry errors. Thus, form design should take into consideration variance in user behavior (frequent vs. infrequent users; expert vs. novice users) in order to maximize efficiency, and de-facto standards, using what users already know and understand to prevent errors in data entry.

The different theoretical and practical considerations mentioned above can be summarized into two overarching guidelines:

(1) Make required fields clearly visible on first sight (Fowler et al., 2004; Horton, 2005; Koyani et al., 2004; Shneiderman and Plaisant, 2004; Tidwell, 2005; Wroblewski, 2008) in order to facilitate fill-in for novice users and

speed up perception by expert users. An asterisk is probably not the preferred visual mark because it takes more time to perceive (Ware, 2008), although it can be read by a screenreader (Fowler et al., 2004).

(2) Add additional visual elements if the user wants to leave the context and a required field is still empty, in order to draw his attention to the correct location on the screen (Apple Inc., 2008). This supports the switch from completion mode to revision mode (Bargas-Avila et al., 2007) and is modeless (Cooper et al., 2007).

This study is concerned with the overarching guideline 1 mentioned above, namely the marking of required fields. The aim of the study is to explore an alternative to the visually not very salient asterisk next to field labels: Marking required fields by coloring their background, a measure that should ease form fill-in for users according to recommendations to clearly mark required fields. In particular, professional users of a financial services firm were asked to fill out a rather complex online form in a real-life task. The number of errors they committed was measured as was the speed with which they could complete the form when the required fields of the online forms were either marked by an asterisk, as is the convention in web design (Wroblewski, 2008), or colored. In this study, the following questions are explored: Does marking required fields with background color instead of using an asterisk besides the text label lead to fewers errors in form fillin? Are participants able to complete form fill-in faster if required fields are marked with background color instead of asterisks? And finally, does marking required fields with colored backgrounds affect user satisfaction positively compared to asterisk markings?

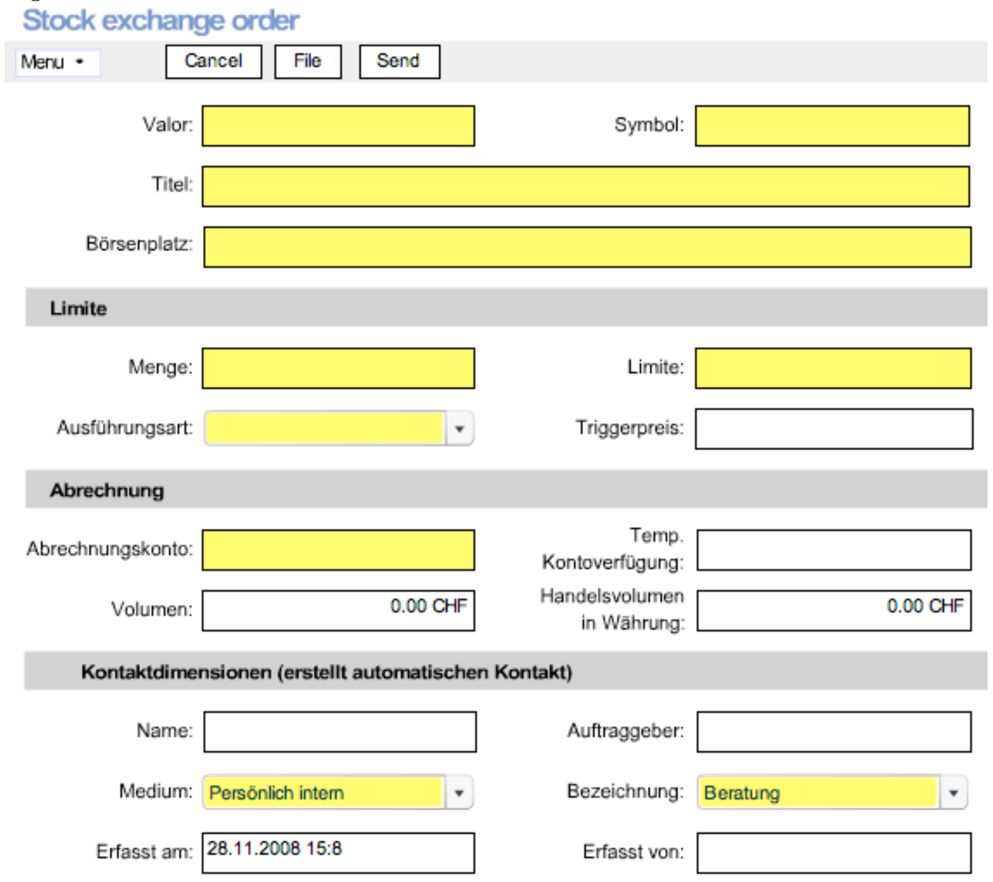
2 Method

2.1 Design

For this study, participants filled out two different versions of a form that is part of a browser-based CRM application. The study used a related samples design. The independent variable was the type of marking applied to required fields. It had two levels: Marking by labels with asterisks and marking by colored field backgrounds. Figures 1 and 2 show parts of the forms with different marking types for required fields.

Dependent variables were the number of errors a participant made during the task, task completion time and a post-test questionnaire to assess a participant's satisfaction with the user interface. Once a participant tried to submit a form, each empty required field was counted as an error. A message was presented to the user, containing information about the missing required fields.

Fig. 1. Part of the stock exchange order form with required fields marked by colored backgrounds.



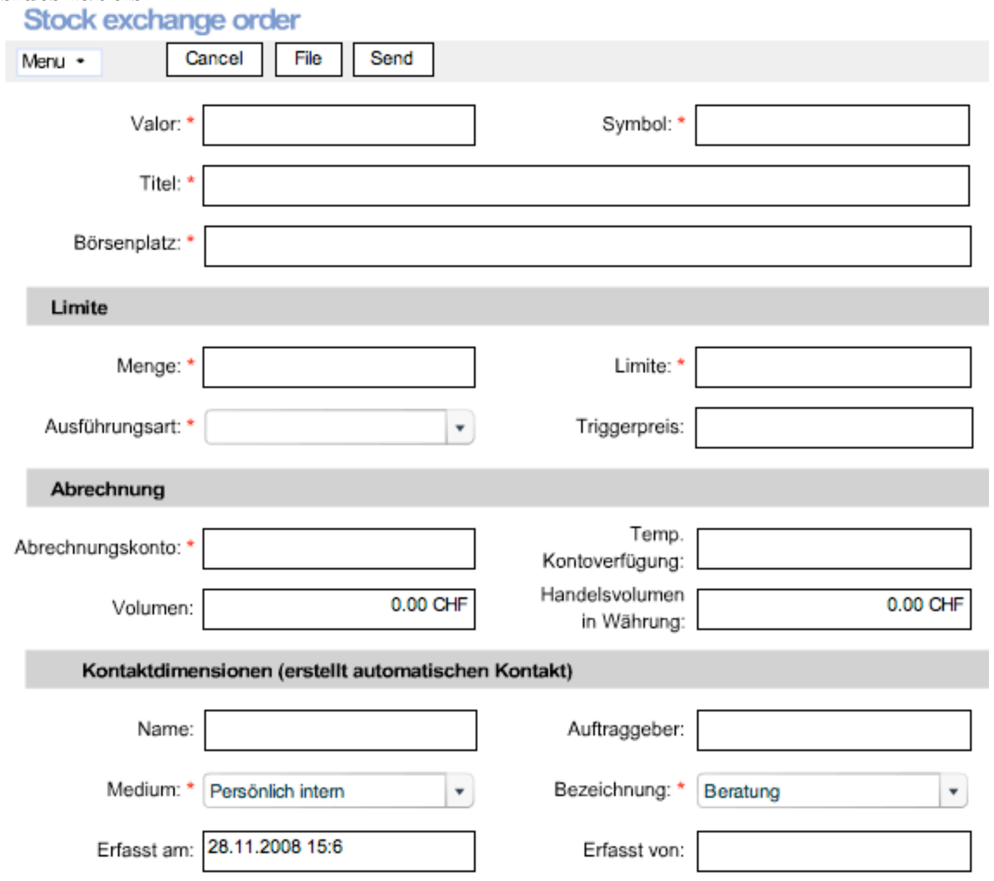
Valor:		Symbol:	
Titel:			
Börsenplatz:			
Menge:		Limite:	
Ausführungsart:		Triggerpreis:	
Abrechnungskonto:		Temp. Kontoverfügung:	
Volumen:	0.00 CHF	Handelsvolumen in Währung:	0.00 CHF
Name:		Auftraggeber:	
Medium:	Persönlich intern	Bezeichnung:	Beratung
Erfasst am:	28.11.2008 15:8	Erfasst von:	

The form was not submitted until all required fields were complete. The short form of QUIS (Chin et al., 1988), a validated and widely used questionnaire (Shneiderman and Plaisant, 2004) for user interface satisfaction, was applied to measure user satisfaction.

2.2 Participants

The study was conducted with 24 participants. Their age ranged from 21 to 48 with a mean of 32 ($SD = 7.2$). Thirteen of the participants were women. All participants were employees of a financial institute. They were all users of the CRM application from which the manipulated forms were taken. The CRM application implemented both versions of required field markings in different screens in an inconsistent manner, therefore participants were used to both variants.

Fig. 2. Part of the stock exchange order form with required fields marked by asterisks besides labels.



Valor: *
Symbol: *
Titel: *
Börsenplatz: *

Limite

Menge: *
Limite: *
Ausführungsart: *
Triggerpreis:

Abrechnung

Abrechnungskonto: *
Temp. Kontoverfügung:
Volumen: 0.00 CHF
Handelsvolumen in Währung: 0.00 CHF

Kontaktdimensionen (erstellt automatischen Kontakt)

Name:
Auftraggeber:
Medium: * Persönlich intern
Bezeichnung: * Beratung
Erfasst am: 28.11.2008 15:6
Erfasst von:

2.3 *Apparatus and Materials*

The two versions of the web based form were recreated as part of a mock-up of the CRM application using HTML and Adobe Flash technology. Flash was used so the look and feel of the proprietary CRM application could be mimicked. The mock-up was presented on a laptop computer and sessions were recorded with the usability test recording software TechSmith Morae (version 2.0.1). Errors were tracked using markers in Morae. Task completion time was logged automatically by the software. Participants received a paper interview guide that included a short demographic questionnaire, the instructions to the two tasks they had to complete, and the short version of QUIS after every task. Finally, two additional questions where added, to explore whether the experimental task was realistic and how frequently the participants encounter it during their work.

2.4 Procedure

Participants completed the experiment individually. They were presented with the start page of the application mock-up and were told to complete the questions and tasks in the interview guide. The two tasks in the interview guide had the following form:

"Customer <C> called and asked you to buy <N> registered shares of <Stock X> (Symbol: <Y>) at market. Use portfolio number <Z>."

After completing the task, all participants answered the satisfaction questionnaire and the task suitability questions. Every participant completed two tasks, one for every required field marking type. The order of the two required field marking types was alternated to counter sequence effects: Half the participants' forms had required fields marked by asterisks during the first task and colored backgrounds during the second task, the other half of the participants had forms with required fields marked by colored backgrounds during the first task and asterisks during the second task.

3 Results

The measured dependent variables of the two required field marking conditions are shown in table 1.

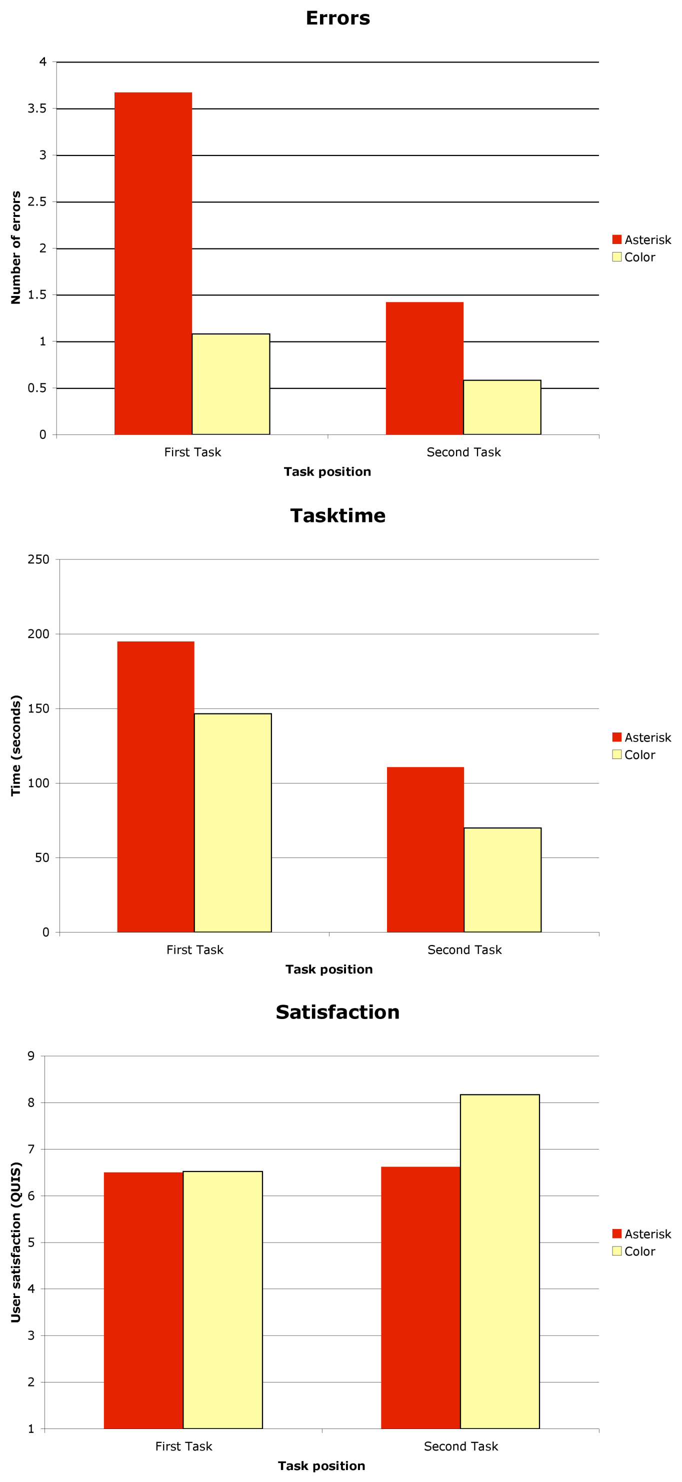
Table 1
Statistical parameters for different required field markings.

		Asterisk	Colored background
Measures	n	*M* (*SD*)	*M* (*SD*)
Number of errors	24	2.540 (2.621)	.830 (1.239)
Task completion time (s)	24	152.852 (96.008)	108.146 (71.912)
Satisfaction (QUIS)	24	6.560 (1.454)	7.342 (1.775)

To test the hypotheses that colored backgrounds as required field markings lead to fewer errors, faster form fill-in and greater satisfaction compared to asterisk markings, t-tests for related samples and an alpha level of .05 were used. There were statistically significant differences: Marking required fields by colored background caused fewer errors, $t(23) = 2.777$, $p = .006$, shorter task completion time, $t(23) = 1.836$, $p = .04$, and higher satisfaction, $t(23) = 1.754$, $p = .047$ (one-tailed tests).

To control the applied sequence effect counter measures, the data were analysed using two-way analyses of variance (ANOVA) with required field marking

Fig. 3. Average numbers of errors, task completion times and satisfaction ratings for different required field markings split by task position.



(asterisk, colored background) and experimental sequence (asterisk - colored background, colored background - asterisk) as the two factors and an alpha level of .05. Dependent variables were, again, number of errors, tasktime and satisfaction. Figure 3 shows that in the second trial, participants committed fewer errors, $F(1,\ 22) = 6.102$, $p = .022$, were faster, $F(1,\ 22) = 19.844$, $p < .001$, and were more satisfied, $F(1,\ 22) = 4.510$, $p = .045$, no matter what task they did second. Alternating the order of required field markings in the tasks successfully countered these sequence effects as all three analyses showed no significant main effect for experimental sequence for number of errors, $F(1,\ 22) = 2.538$, $p = .125$, tasktime, $F(1,\ 22) = .023$, $p = .881$, and satisfaction $F(1,\ 22) = 2.631$, $p = .119$.

4 Discussion

Participants completed form fill-in commiting significantly fewer errors when required fields were indicated by colored backgrounds rather than by the usual asterisk. Furthermore, the colored background let them complete the forms significantly faster and more satisfied.

Since we measured task completion time as the duration from initial display of the form to its succesfull completion, we have to assume that task times were influenced directly by whether a form could be submitted free of errors on the first try or whether the participants had to deal with it again after an unsuccessful attempt. The influence of task completion times was therefore to a large degree moderated by the number of errors committed. This had the advantage of getting a realistic picture of the actual time loss during form fill-in from the users perspective: Complete successfull form fill-in takes 41% more time if required fields are not properly highlighted. It was also shown that, although a general performance increase from first to second form fill-in exists, this benefit is in addition to any requiredness-marker related differences. In other words, colored backgrounds are better markers for requiredness than asterisks no matter how well a form is known.

An important question is why required fields with colored backgrounds help participants to commit fewer errors? Although the asterisk is a de-facto standard to indicate required fields, it seems the additional saliency provided by the colored background, which is of course much larger than the asterisk, has a much better chance of preventing incomplete form submission. The fact that participants were significantly more satisfied with the forms with colored required fields suggests that the raised saliency is a welcome help rather than an annoying or unnecessary distraction.

The practical implications of this study are therefore that form designers can ease form input for users by marking required fields with a colored background. One disadvantage of colored required fields was mentioned earlier: The inability of screenreaders to recognize the color-markings. Consequently we advise designers to use colored backgrounds as an additional marker of requiredness rather than an exclusive one.

However, this study does however not answer the question, to what degree this finding is applicable to other forms. The form used in this experiment was quite complex and targeted towards a professional completing a critical task. Many forms on the Internet (e.g. forms for account creation) are smaller and have fewer or less dire consequences in the event of errors than an erroneous stock exchange order. However, many online forms are appearing in various order and checkout process pages where the support provided by improved required field marking could prove helpful. It should be mentioned, that it is not possible to draw any conlusions on what color should be used to highlight the required fields' backgrounds. Yellow backgrounds provided the results reported here, but different factors could influence the best color choice. Usage of colors throughout a page's or application's design could influence saliency of yellow required fields as well as general or site-specific information or warnings implied by the use of certain colors.

Acknowledgements

The authors would like to thank the Zürcher Kantonalbank (ZKB) in Zürich, Switzerland for the support and funding of this research as part of the UCD ZKBconnect project.

References

Apple Inc., 2008. Apple human interface guidelines: User experience.
URL `http://developer.apple.com/documentation/UserExperience/Conceptual/AppleHIGuid`

Bargas-Avila, J. A., Oberholzer, G., Schmutz, P., de Vito, M., Opwis, K., 2007. Usable error message presentation in the world wide web: Don't show errors right away. Interacting with Computers 19 (3), 330–341.

Baxley, B., 2002. Making the Web Work: Designing Effective Web Applications. Sams Publishing.

Chin, J. P., Diehl, V. A., Norman, K. L., 1988. Development of an instrument measuring user satisfaction of the human-computer interface. In: CHI '88: Proceedings of the SIGCHI conference on Human factors in computing systems. ACM, New York, NY, USA, pp. 213–218.

Cooper, A., Reimann, R., Cronin, D., 2007. About Face 3: The Essentials of Interaction Design. John Wiley & Sons.

Fowler, S., Stanwick, V., NetLibrary, I., 2004. Web Application Design Handbook: Best Practices for Web-based Software. Morgan Kaufmann.

Horton, S., 2005. Access by Design: A Guide to Universal Usability for Web Designers. New Riders Publishing Thousand Oaks, CA, USA.

Klossek, U. M. H., Russell, J., Dickinson, A., 2008. The control of instrumental action following outcome devaluation in young children aged between 1 and 4 years. Journal of Experimental Psychology: General 137 (1), 39–51.

Koyani, S. J., Bailey, R. W., Nall, J. R., Allison, S., Mulligan, C., Bailey, K., Tolson, M., 2004. Research-Based Web Design and Usability Guidelines. Washington, DC: Health and Human Services Department.

Nielsen, J., 1993. Usability Engineering. Academic Press.

Nielsen, J., 1994. Enhancing the explanatory power of usability heuristics. In: Proceedings of the SIGCHI conference on Human factors in computing systems. pp. 152–158.

Nielsen, J., Landauer, T., 1993. A mathematical model of the finding of usability problems. Proceedings of the SIGCHI conference on Human factors in computing systems, 206–213.

Polson, P. G., Lewis, C. H., 1990. Theory-based design for easily learned interfaces. Human-Computer Interaction 5, 191–220.

Shneiderman, B., Plaisant, C., 2004. Designing the User Interface: Strategies for effective Human-Computer Interaction, 4th Edition. Addison-Wesley.

Taylor, F. W., 1911. The Principles of Scientific Management. Harper & Brothers Publishing.

Tidwell, J., 2005. Designing interfaces: patterns for effective interaction design. O'Reilly.

Tullis, T. S., Pons, A., 1997. Designating required vs. optional input fields. In: CHI '97: CHI '97 extended abstracts on Human factors in computing systems. ACM, New York, NY, USA, pp. 259–260.

US Federal Bureau of Justice Assistance, 2008.
URL `http://www.ojp.usdoj.gov/BJA/evaluation/glossary/`

Ware, C., 2008. Visual Thinking for Design. Morgan Kaufman.

Wroblewski, L., 2008. Web Form Design: Filling in the Blanks. Rosenfield Media.

The organization of interaction design pattern languages alongside the design process

Christian Hübscher, Stefan L. Pauwels, Sandra P. Roth,
Javier A. Bargas-Avila, Klaus Opwis

University of Basel, Department of Psychology,
Center for Cognitive Psychology and Methodology, 4055 Basel, Switzerland

Abstract

This work explores the possibility of taking the structural characteristics of approaches to interaction design as a basis for the organization of interaction design patterns. The *Universal Model of the User Interface* (Baxley, 2003) is seen as well suited to this; however, in order to cover the full range of interaction design patterns the model had to be extended slightly. Four existing collections of interaction design patterns have been selected for an analysis in which the patterns have been mapped onto the extended model. The conclusion from this analysis is that the use of the model supports the process of building a pattern language, because it is predictive and helps to complete the language. If several pattern writers were to adopt the model, a new level of synergy could be attained among these pattern efforts. A concluding vision would be that patterns could be transferred freely between pattern collections to make them as complete as possible.

Keywords: design patterns; pattern languages; interaction design

1 Introduction

In the project that was the trigger for this research (see Pauwels et al., 2009, 2010) one of the challenges was to build a library of interaction design patterns for an internal system. This library had to be designed to cover the whole design space of this application with patterns. Then it had to be positioned as an authoritative source of information about interaction design for the business analysts and developers in the company. This is the background of this work. The article now explores whether it is possible to take knowledge from approaches to interaction design as a basis for the organization of interaction design patterns. Publicly available pattern collections (Tidwell, 2006; van Duyne et al., 2007; Yahoo! Inc., 2009; van Welie, 2009) are used to illustrate the analysis.

1.1 The problem of pattern categorization

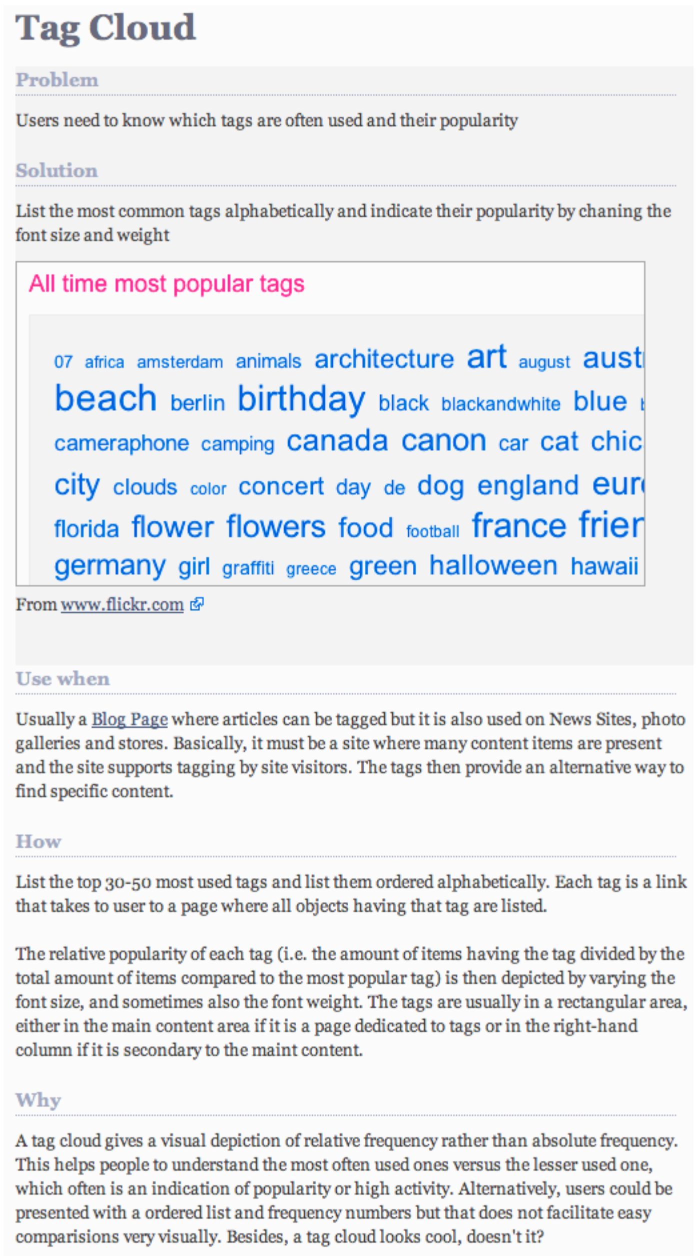
In recent years, many interaction design pattern collections have been published and more are appearing each year (for an overview, see: http://www.hcipatterns.org). With collections growing bigger, the question of pattern categorization becomes more important. According to Dearden and Finlay (2006), the organization of pattern languages is an important area of research in human-computer interaction (HCI).

There is currently a certain "duplication of effort" (Dearden and Finlay, 2006, p. 88) in the field of interaction design patterns and this indicates that in building pattern languages, the wheel has already been (re-)invented several times. Ironically, this is exactly what the concept of patterns intends to prevent designers from doing. Because pattern collections are all organized differently, it is very hard to compare them and to transfer individual patterns from one collection to the other. With the *pattern language markup language* (PLML) (Fincher, 2003), important steps have been made toward a standardization of the structure of individual patterns, but no such organized effort has yet been made to find a unified organization of pattern languages. To have such a standard in the *organization of languages* may bring synergies to the HCI field as a whole and make it easier for individual projects to build their own pattern language, based on the work of others.

1.2 Interaction design patterns and pattern languages

Dearden and Finlay (2006) define a pattern as "a structured description of an invariant solution to a recurrent problem within a context" and a pattern language as "a collection of such patterns organized in a meaningful way". The concept of design patterns was originally developed by Christopher Alexander (1964) in the field of architecture. Software engineers Gamma et al. (1995) then adopted the concept to describe software design patterns (see Gabriel (1996) for more about these efforts). Borchers (2001) later applied patterns to problems in the field of HCI. One emerging area of an application for design patterns, which is closely related to HCI, is e-learning (see Dimitriadis et al., 2009).

This study focuses on *interaction design patterns*: “A problem is stated in the domain of human interaction issues, and the solution is stated in terms of suggested perceivable interaction behavior” (Dearden and Finlay, 2006, p. 52). The description of a pattern is structured with the topics *problem*, *context*, *solution*, and *forces* included in many cases; however, the structure and naming of these sections vary among pattern collections (see Figure 1). Most include further sections such as examples.



Tag Cloud

Problem

Users need to know which tags are often used and their popularity

Solution

List the most common tags alphabetically and indicate their popularity by chaning the font size and weight

All time most popular tags

07 africa amsterdam animals architecture art august aust
beach berlin birthday black blackandwhite blue
cameraphone camping canada canon car cat chic
city clouds color concert day de dog england eur
florida flower flowers food football france frier
germany girl graffiti greece green halloween hawaii

From www.flickr.com

Use when

Usually a Blog Page where articles can be tagged but it is also used on News Sites, photo galleries and stores. Basically, it must be a site where many content items are present and the site supports tagging by site visitors. The tags then provide an alternative way to find specific content.

How

List the top 30-50 most used tags and list them ordered alphabetically. Each tag is a link that takes to user to a page where all objects having that tag are listed.

The relative popularity of each tag (i.e. the amount of items having the tag divided by the total amount of items compared to the most popular tag) is then depicted by varying the font size, and sometimes also the font weight. The tags are usually in a rectangular area, either in the main content area if it is a page dedicated to tags or in the right-hand column if it is secondary to the maint content.

Why

A tag cloud gives a visual depiction of relative frequency rather than absolute frequency. This helps people to understand the most often used ones versus the lesser used one, which often is an indication of popularity or high activity. Alternatively, users could be presented with a ordered list and frequency numbers but that does not facilitate easy comparisions very visually. Besides, a tag cloud looks cool, doesn't it?

Figure 1. Example of an interaction design pattern description (van Welie, 2009)

Alexander’s (1964, 1979) idea behind design patterns is (see Kohls and Uttecht, 2009) that they should support design as a problem-solving task to achieve fitness between form and context. A design problem occurs because competing “forces” have to be satisfied. Thus multiple design patterns can solve the same problem in different contexts (e.g. *checkboxes* and a *list builder* both solve the problem of selecting multiple items). In this case, they have identical problem attributes but the context attributes make clear when to choose one pattern over the other. Alexander uses the term *forces* to describe these context-dependent constraints that have an effect on how to solve the problem. The proper configuration of a group of patterns is in itself a pattern on a more abstract level. Alexander stresses the point that one needs a *pattern language* to achieve a coherent design and that “a bunch of good ideas” (Alexander, 1999, p. 75) are not enough to do the job. Pattern languages should be capable of *producing* coherent wholes; i.e. they should be *generative*.

Dearden and Finlay (2006) conclude that there are two evident forms of organization in Alexander et al. (1977): patterns come in sets according to *levels of physical scale* and build up a *network*, where patterns are referenced to other patterns. The linking of individual interaction design patterns is usually made in a “related patterns” section, where alternative solutions to similar contexts or patterns are placed. Van Welie and van der Veer (2003) distinguish between three fundamental relations:

- *Aggregation*: A design pattern can include others that complete it.
- *Specialization*: A design pattern can be derived and specialized from another design pattern.
- *Association*: Multiple design patterns can occur in the same context or solve similar problems.

An interesting question concerns the *completeness* of a pattern language. Alexander (1979) argues that a pattern language can be morphologically and functionally complete: It is *morphologically* complete when it can account for a complete design, without any missing parts, and *functionally* complete when it is self-consistent, i.e. it does not create forces that it cannot resolve. For interaction design patterns, van Welie and van der Veer (2003) say that a pattern language is complete when every good design that we find can be described using it.

1.3 Recent work on the organization of interaction design patterns

In architecture, Alexander (1979) defined *physical scale* as being the main organizational principle for patterns. The organization of interaction design patterns, on the other hand, is not so straightforward; therefore, we must put more effort into coming up with good organizational principles for patterns in the field of HCI. Different approaches on how to work out an organization of interaction design patterns have been suggested (for an overview, see Dearden and Finlay, 2006). Several authors argue that the best way to organize a pattern language is alongside a design process. Fincher and Windsor (2000) discuss different organizing principles. Their final solution brings their taxonomies in an order that could be associated with the phases of a design process. They distinguish: *analysis space* (context and values), *problem space* (structure: tasks; structure: information) and *solution space* (structure: scale). Van Welie and van der Veer (2003) argue that the organization should be based on a top-down design process and they distinguish the following levels: *business goals*, *posture level*, *experience level*, *task level*, and *action level*. These authors' organization is done according to a design process but they do not explicitly relate it to concrete user-centered or interaction design approaches. They only mention Cooper et al. (2007) as a basis for their choice of the category "posture type patterns". Borchers (2000) maps different types of patterns onto Nielsen's (1993) usability engineering lifecycle but he does this more to argue that we can use patterns across a whole project lifecycle than to discuss the organization of patterns. Dearden and Finlay (2006) give an overview of the different requirements for an organizing principle for pattern languages. According to Fincher and Windsor (2000), an organizing principle should *taxonomise*, *proximate*, be *evaluative* and *generative*; i.e. it should enable users to find (related) patterns, it should allow users to consider the problem from different viewpoints and to build new solutions. Fincher (2002) argues that it would also be desirable that an organization is *predictive*; i.e. it should actively support the process of identifying new patterns. Using the periodic table of the elements in chemistry as an example; she argues that such an organization would help to discover missing patterns. This is a very interesting idea, which suggests giving preference to a top-down categorization based on a certain model over a bottom-up approach.

2 Structural characteristics of interaction design approaches

As mentioned above, several authors suggest that the organization of interaction design patterns should be based on a design process (e.g., van Welie and van der Veer, 2003; Fincher and Windsor, 2000). There is a vast literature on the topic of how to proceed in designing user interfaces, and many design processes have been published so far. This study seeks to extract the relevant aspects from these published works and use them as a basis for structuring interaction design pattern languages. The focus is on the *design* aspect of such processes – even though patterns can also be used to support other phases of the process (see Borchers, 2000, or Granlund et al., 2001).

Different sources can be considered as *interaction design processes;* i.e. usability engineering lifecycles or user-centered design processes:

- Nielsen's Usability Engineering Lifecycle (Nielsen, 1993)
- Delta Method (Rantzer, 1996)
- Contextual Design (Beyer and Holtzblatt, 1998)
- OVID (Robert, 1998)
- Mayhew's Usability Engineering Lifecycle (Mayhew, 1999)
- Usage Centered Design (Constantine and Lockwood, 1999)
- The Elements of User Experience (Garrett, 2002)
- Universal Model of a User Interface (Baxley, 2002, 2003)
- Goal Directed Design (Cooper et al., 2007)

To define how to organize interaction design patterns alongside a design process, we extract the structural characteristics of these approaches in order to perform a mapping of different kinds of patterns onto them. Most approaches to interaction design foster a layered approach in which different levels of the user interface are designed one after another. Some approaches distinguish two phases; others have more levels of design.

Many design approaches make the distinction between *conceptual* and *concrete* design, or conceptual and physical design as Sharp et al. (2007) call it. Rantzer (1996), Beyer and Holtzblatt (1998), Robert (1998), Constantine and Lockwood (1999), and Cooper et al. (2007) make such a distinction in the processes that they describe. Mayhew (1999) distinguishes a 1st, 2nd, and 3rd level of design in her process, but in her levels 2 and 3 concrete design is carried out. In level 2, the central and recurring interactions are designed and in level 3 the rest of the user interface is specified. Here, her distinction is more a matter of scope than a matter of different aspects. One can argue that Mayhew's design process is also based on the distinction of *conceptual* and *concrete* design.

In *conceptual design,* the structural base of the user interfaces – the "user interface architecture" – is defined. The name for this design task varies: conceptual design (Rantzer, 1996), user environment design (Beyer and Holtzblatt, 1998), conceptual model design (Mayhew, 1999), content model (Constantine and Lockwood, 1999), and interaction framework (Cooper et al., 2007). In this phase, the designer works out relationships between user objects, organizational schemes, and workflows. The deliverables of these tasks are, for the most part, diagrams, storyboards, and sketches of user interfaces. Some of these deliverables do not really look like user interfaces.

In *concrete design,* the user interface – in the form of concrete user interface elements – is defined. This task is called prototyping (Beyer and Holtzblatt, 1998; Rantzer, 1996), detailed design (Cooper et al., 2007; Mayhew, 1999), or the implementation model (Constantine and Lockwood, 1999). The deliverables of concrete design are interactive prototypes or renderings of screens, which often look and behave very similarly to the real system.

Besides the distinguishing of two phases of design, there are several authors who describe an approach with three or more levels. These approaches, however, are not contradictory to the notion of *conceptual* and *concrete* design; they are more an extension of this thinking. The classical work of IBM (1992) explains the levels of designing a user interface with the metaphor of an iceberg, which has the three levels: *structure*, *behavior*, and *presentation*. The approaches by Garrett (2002) and Baxley (2003) build on these levels. Garrett (2002) has a model with the five layers: *strategy*, *scope*, *structure*, *skeleton*, and *surface*. The layers *strategy* and *scope*, however, are more to "set the stage" for doing the interaction design, although *structure*, *skeleton*, and *surface* are similar to the layers of the aforementioned "iceberg model". The tiers of *structure*, *behavior*, and *presentation* bring in a more sophisticated discrimination between different types of design tasks and hence patterns. Following this thinking, using website navigation as an example (see e.g., Leuthold et al., 2011), one can distinguish patterns that describe the *structure* of navigation (e.g., hierarchical vs. flat), the *behavior* of navigation (e.g., dynamic vs. static), and the *presentation* of navigation (e.g., left vs. horizontal placement in the layout). All these different aspects of navigation are influenced by their own forces and therefore it makes sense to distinguish between these aspects by using different interaction design patterns.

An elaborate model in this sense is Baxley's (2003) *Universal Model of the User Interface*. In his model, the same three main tiers exist as in the "iceberg model": *structure*, *behavior*, and *presentation*. However, these three tiers are further divided into three sub-layers each:

- Structure
  - *Conceptual model*
  - *Task flow* (formerly called *structural model* by Baxley, 2002)
  - *Organizational model*
- Behavior
  - *Viewing and navigation*
  - *Editing and manipulation*
  - *User assistance*
- Presentation
  - *Layout*
  - *Style*
  - *Text*

The models of Baxley (2003) and Garrett (2002) are similar, because they both describe a layered top-down design approach. In general, they can be seen as similarly well suited to organizing patterns into categories, but Baxley's (2003) model is much more fine-grained. It actually distinguishes nine layers that are relevant for interaction design patterns. For this reason, it is taken as a basis for this analysis. However, Garrett's (2002) model has a wider scope and therefore it will be taken as an extension of Baxley's model to cover the whole range of interaction design patterns (see section 3.2).

3 A model for the organization of interaction design patterns

It is a goal of this research to find a model that is based on an interaction design process and that can be used to organize interaction design patterns. As indicated above, Baxley's model has the required properties, so its detailed structure is presented here as described by Baxley (2002, 2003). However, the model does not cover the full range of interaction design patterns as defined by Dearden and Finlay (2006). Therefore the authors have made an extension to the model with the introduction of the category "requirements patterns". Following this, the relationship between the model and technical platforms is discussed.

3.1 The original structure of Baxley's model

Baxley's "Universal Model of the User Interface" (2003) has similarities with the other models described above but it divides design tasks in a more sophisticated way. The model has nine layers divided between three tiers (see Figure 2). Baxley divides these layers into further topics and sub-topics (Baxley, 2002), see Table 1.

The nine layers of Baxley's model distinguish different aspects of a user interface; for example, whether we are dealing with the *structure* of the user interface or with its *behavior* and whether the behavior is for the manipulation of data by users (i.e. *editing and manipulation*) or for helping them by doing so itself (i.e. *user assistance*). These nine layers can be seen as building on each other.

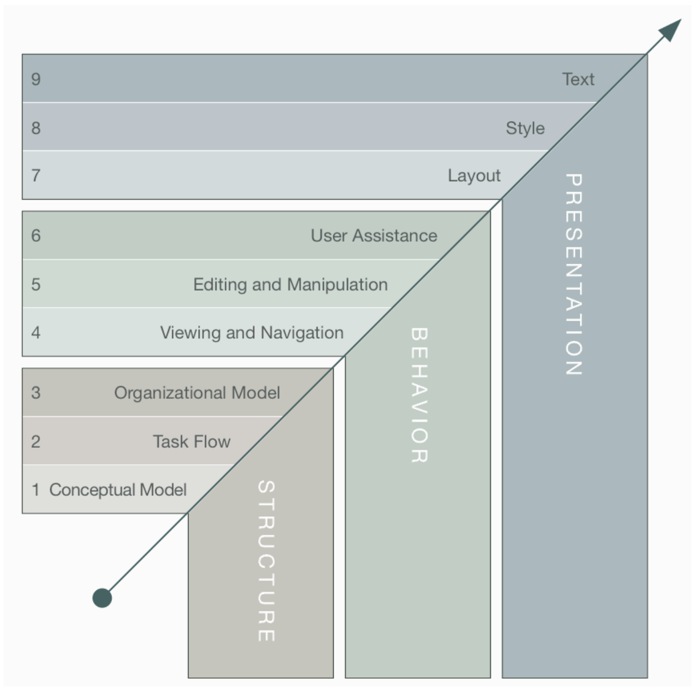


Figure 2. Illustration of Baxley's "Universal Model of the User Interface" (Baxley, 2003)

Baxley (2002) breaks down most of the layers in a topical manner (see the column "topics" in Table 1). These sub-divisions cannot be seen as clearly building on each other. Most of them are topical in nature and are often just different aspects of a layer. This finer structure is optimized for the design of web applications (which is the topic of Baxley's book; Baxley, 2002) and it has been created to provide an optimal structure for the "patterns" that Baxley (2002) discusses. Baxley mentions "interaction design patterns" for all the different levels of the user interface of web applications; however, he does not call them "patterns" but rather "conventions": "Unfortunately, the use of the word 'pattern' in this context, although definitely accurate, is a bit arcane." (Baxley, 2002, p. 14). Baxley seems to have developed the model to organize interaction design patterns – as well as for other purposes – but uses a different terminology.

Table 1. The layers broken down into topics (Baxley, 2002)

Tier	Layer	Topics	Sub-topics
Structure	*Conceptual Model*	"Examples"	Store; Catalog
	Structural Model (later called *task flow* by Baxley, 2003)	Pages	Views; Forms; View/Form Construct
		Workflows	Hubs; Wizards; Guides (hub/wizard hybrid)
	Organizational Model	Classification schemes	Objective: Alphabetic; Numeric; Chronologic; Geographic Subjective: Topical; Functional; Audience-based; Metaphorical
		Models of association	Indexes; Hierarchies; Webs
Behavior	*Viewing and Navigation*	Navigation	High-level navigation; Low-level navigation; Utility navigation
		Selecting objects and issuing commands	Shared controls; Dedicated controls
		Viewing lists of data	Changing column sets; Paging; Sorting; Filtering; Searching
	Editing and Manipulation	Input controls	Check boxes; Radio buttons; List boxes; Menus; Text boxes; Buttons
		Common interaction problems and solutions	Selecting a single item; Selecting multiple items; Selecting a date
	User Assistance	Help	Conceptual help; Procedural help; Definitional help; Instructional help
		Alerts	Error alerts; Status alerts; Confirmation alerts
Presentation	*Layout*	Simplicity	Clarity; Reduction; Leverage
		Consistency	Web conventions; Templates and grids; Standards and guidelines
		Order	Grouping; Hierarchy; Alignment
	Style	Evaluating style	Individuality; Brand consistency; Appropriateness for the audience and function
		Preventing style from messing other things up	Working within the medium; Legibility (contrast, line length, typeface, type size, font styling, density/leading, balance the variables affecting legibility); Providing visual cues to behavior (visual cues for text-based hyperlinks, visual cues for clickable images)
	Text	Eliminate superfluous text	Eliminate superfluous text
		Text: what's it good for	Navigation; Titles; Labels; Instruction and help; Marketing messages
		Writing for the web	Be courteous, not patronizing; Leverage the context; Don't repeat yourself; Avoid multisyllabic words that obfuscate

3.2 An extension of Baxley's model

The definition of *interaction design patterns* is meant to distinguish these patterns from *user interface software design patterns* (Dearden and Finlay, 2006) in the area of patterns concerning the user interface (see Figure 3). The former are concerned with the perceivable aspects of the user interface and the latter with the inner working of the system related to the user interface.

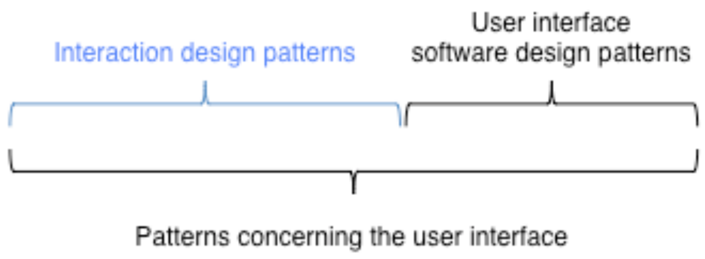

Interaction design patterns
User interface software design patterns
Patterns concerning the user interface

Figure 3. Different patterns concerning the user interface

The perceivable aspects of the user interface, which can be documented as interaction design patterns, can be of two kinds: (a) *user requirements* i.e. a function to enable a user to achieve a certain goal and (b) the *conceptual implementation* of these requirements in the form of a user interface. Baxley's Model is very detailed but does not cover the whole scope of interaction design patterns. The conceptual implementation (*how* it is done) is the scope of Baxley's model, which further breaks down the different solutions into categories. The user requirements (*what* is implemented in the user interface) can be seen as beyond the scope to Baxley's model and suggest an extension of the model (see Figure 4). Baxley describes a requirements phase in his book but does this within his larger scope design process (Baxley, 2002, p. 44):

1) *Understanding* (user needs, competition, business opportunity, technical constraints)
2) *Vision* (core design values, opportunity statement, persona profiles & goals)
3) *Requirements* (functional, technical, business, usability)
4) *Design* (structure, behavior, presentation)

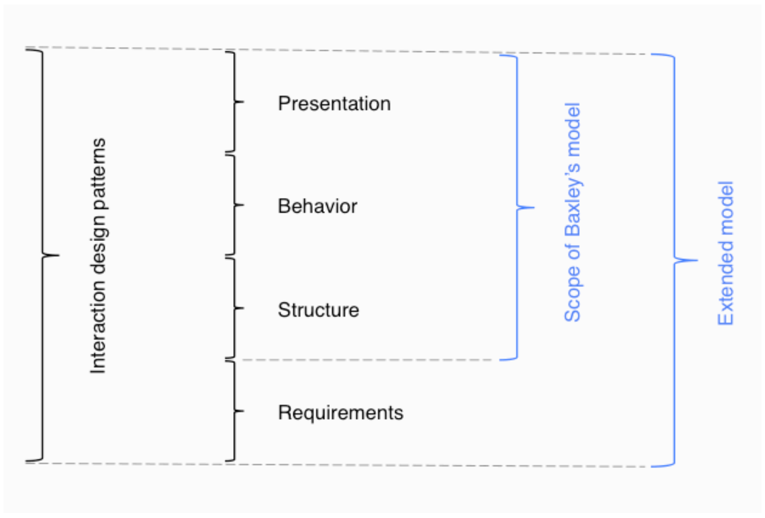


Figure 4. The different interaction design patterns

This extended model bears a similarity to Garrett's (2002) model, which also contains a layer dealing with requirements (see Figure 5). This *scope* layer deals with the question of whether a feature or function is part of a system's functional requirements or not, whereas the *strategy* layer does not deal with *solutions* to users' problems but rather with the definition of the *needs of users* or the *business objectives* and therefore is beyond the scope of interaction design patterns.

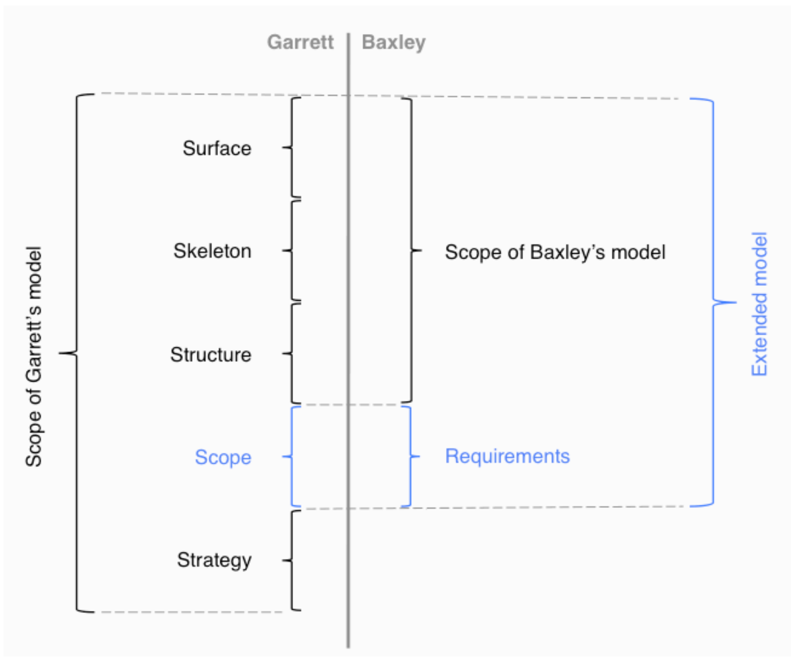


Figure 5. Comparison of the models of Garrett (2002) and Baxley (2003)

Garrett's (2002) *structure*, *skeleton*, *surface* and Baxley's (2003) *structure*, *behavior*, *presentation* both cover the conceptual implementation of the user interface but they have certain differences in the mapping of patterns onto the model; for example, Baxley (2003) puts "layout" in the *presentation* tier whereas Garrett (2002) sees it as part of his *skeleton* layer (which otherwise would correspond more to *behavior*). For the sake of this analysis, however, it is not relevant to analyze these differences to a further extent.

These "requirements patterns" and "patterns for the conceptual implementation of a user interface" are described below. The definitions are meant to distinguish between these two types of patterns. It is important so separate them accurately otherwise this extended model will not work as an organizational model.

3.2.1 Requirements Patterns

Patterns describing abstract features that allow the user to achieve a certain goal are what we call "requirements patterns". In the example of a fictitious "publish to Twitter" function, the pattern would focus on the goal of the user (let friends know of discoveries made while surfing). This pattern would focus on why a user needs a "publish to Twitter" function as opposed, say, to a "subscribe to RSS feed" function. These patterns focus on the forces that distinguish the different goals a user could have, but they do not describe the way that the user achieves these goals in the form of an interaction. This would be described independently of the conceptual and technical implementation. The forces are described on the level of a goal that a user wants to achieve as opposed to another goal. The patterns aim at the optimization of *utility* (Grudin, 1992). If such patterns are described independently of their implementation, the patterns are valid under different circumstances and on different technical platforms. If, on the other hand, the feature is described as a pattern "Twitter icon", the user requirement is mixed up with the conceptual implementation and the solution is no longer the best if circumstances change. In a situation in which multiple such functions (for Twitter, Facebook, etc.) have to be provided for a certain object, the function would no longer be implemented as an icon but would maybe rather be part of a menu. Thus described in an implementation-agnostic way, the requirement pattern "publish to Twitter" would be "stable" under various circumstances.

The patterns discussed above are functional requirements but there are also patterns that describe non-functional requirements. The pattern "site accessibility" (van Duyne et al., 2007) is such an example. It is a very high-level pattern, which might also contain several more detailed patterns (e.g. "hidden jump to navigation link" for users with screen readers, good contrast, the use of certain HTML tags). So the category of "requirements patterns" is meant to include both functional and non-functional requirements that have an impact on perceived aspects of the user interface.

3.2.2 Patterns for the conceptual implementation of a user interface

The patterns for the conceptual implementation of a user interface describe ways to realize user requirements on a conceptual level. These patterns are the different parts used for the implementation of a user requirement. Which of these parts does this best depends on the specific circumstances. These patterns focus on the forces, which are influenced by different configurations of such patterns. If a function is the only function to be used in a list of objects, this function can be conceptually implemented as an icon. If this function is one among many others it might be implemented as a menu. These patterns do not focus on “specific end goals” of users but more on “generic sub-goals”. The patterns aim at the optimization of one aspect or more of *usability* – the effectiveness, efficiency, or satisfaction of the user (Hornbaek, 2006) – but not on the *utility* (Grudin, 1992). With the help of Baxley’s model, the different solutions for the conceptual implementation can be broken down into nine categories.

3.3 The scope of platform applicability of Baxley’s model

Because Baxley (2002) introduced his model to explain “how to make the web work”, one might ask whether this model is also valid for other platforms. Baxley later started to call it the “Universal Model of the User Interface” and discussed ATMs, DVD menu systems, Amazon.com and Microsoft Word to support this point (Baxley, 2003). The fact that IBM (1992) used the same layers of design in a pre-web area shows that at least the main tiers *structure*, *behavior*, and *presentation* are relevant beyond the web. But one might ask, where do the patterns for mobile or rich web interaction design belong? To explain this, it is better to look at the relationship between the layers of Baxley’s model and the technical platform.

In the logic of the model, different platforms can be seen as orthogonal to the tiers *structure*, *behavior*, and *presentation* (see Figure 6). Tidwell’s (2006) patterns are more or less platform-agnostic and can be used on several platforms. Her “one-window drilldown” pattern works on different platforms and she uses examples from the iPod, Mac OS X, and a character-based e-mail application to explain the pattern. The platform-independent description of the pattern and also the implementations on the different platforms unambiguously belong to the *structure* tier. Although her pattern “movable panels” is not so platform-independent it nevertheless belongs to the *behavior* tier. The pattern “expanding screen width” (van Duyne et al., 2007) is rather web-specific but it applies to both “classic” and “rich” websites and belongs to the *presentation* tier. On the other hand, the pattern “self-healing transition” (Yahoo!, 2009) is a pattern from the *behavior* tier (*user assistance*, because it helps the user to better understand the effects of a manipulation) and is targeted at rich interaction web interfaces. It cannot be used in “classic” web applications, because of technical limitations, even though this distinction will fade away more and more with the establishment of new web standards. However, the “self-healing transition” could also be used on a desktop OS or a modern mobile OS.

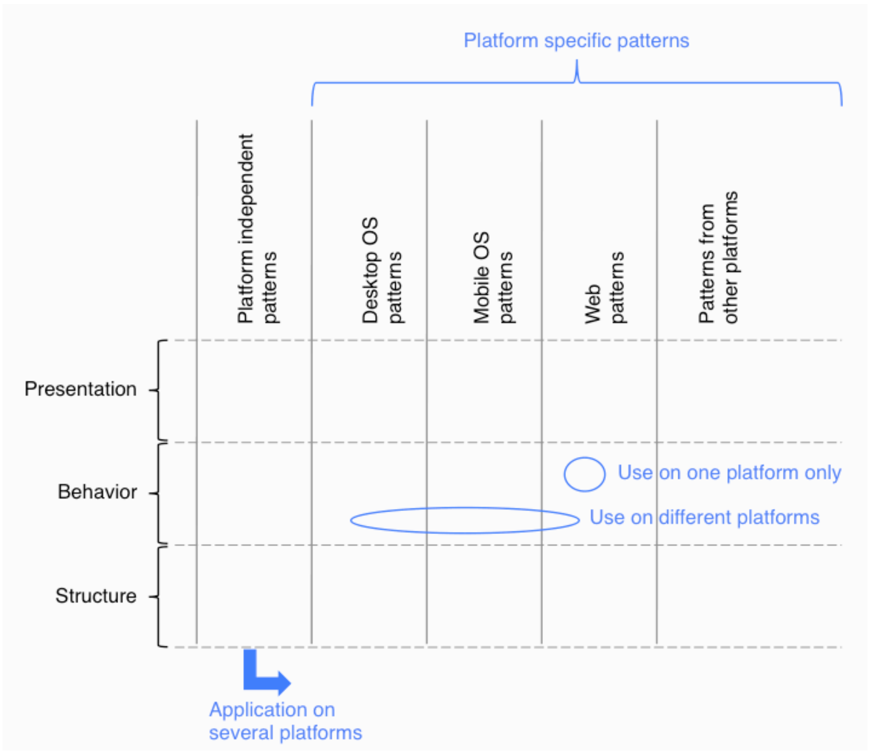

Platform specific patterns
Platform independent patterns
Desktop OS patterns
Mobile OS patterns
Web patterns
Patterns from other platforms
Presentation
Behavior
Structure
Use on one platform only
Use on different platforms
Application on several platforms

Figure 6. Pattern collections for different platforms

These examples show that there are whole pattern collections that are rather platform-independent (e.g. Tidwell’s, 2006). However, in collections written for special platforms there are also patterns that are more or less platform-independent whereas others make no sense in another environment. Aside from the question of platform, all the patterns concerned with the conceptual implementation of the user interface can be categorized into Baxley’s model in a stable way.

4 The analysis of four pattern collections

Following the presentation of Baxley's model, we discuss some interaction design pattern collections in relation to the following eleven categories (ten for interaction design patterns):

- *Requirements* (with an impact on perceived aspects of the user interface)
- *Conceptual Model*
- *Task Flow*
- *Organizational Model*
- *Viewing and Navigation*
- *Editing and Manipulation*
- *User Assistance*
- *Layout*
- *Style*
- *Text*
- *Software Design* (user interface software design patterns)

Because there are dozens of interaction design pattern collections (some published as books but most of them available in the World Wide Web), the analysis focuses on a small sub-set:

- Book: *Designing Interfaces* by Jenifer Tidwell (2006)
- Book: *The Design of Sites* by van Duyne et al. (2007)
- Website: "Welie.com" by Martijn van Welie (2009)
- Website: "Yahoo! Design Pattern Library" by Yahoo! Inc. (2009)

These four interaction design pattern collections have been chosen for this analysis because on the one hand, they contain a certain number of patterns similar in scope but on the other hand, they are well established; i.e., it is likely that they will still be around in the future. In 1997, Tidwell started with an online pattern language called "Common Ground" (Tidwell, 2009) and then, based on that work, published the book *Designing Interfaces* (Tidwell, 2006), making it the pattern book with the longest traceable history. The book *The Design of Sites* (van Duyne et al., 2007) has already been published in its second edition. The first edition dates back to 2002 (van Duyne et al., 2002), which makes it the first fully-fledged collection of interaction design patterns available in the form of a book. The roots of the website "Welie.com" by Martijn van Welie (2009) date back at least to 2000 (see van Welie and Trætteberg, 2000) thus it can be said that it is one of the most established online collections of interaction design patterns. The Yahoo! Design Pattern Library (Yahoo! Inc., 2009) is the most recent collection in this analysis, and contains the least number of patterns. However, the fact that it is the only corporate collection of interaction design patterns that is at least partially public makes it an interesting candidate for this analysis. There is also a case study available for the Yahoo! Pattern Library (Leacock et al., 2005).

4.1 The process of categorization

In the analysis of these four collections, all the patterns were put into the proposed categories in order to explore whether:

- all patterns can be classified into the categories
- there are layers that do not have any patterns in them
- the distribution of patterns across the layers is even or not
- there is any ambiguity in classifying patterns in such a way

The first author conducted the analysis of the four pattern collections. It was identified, for each pattern, on which layers its forces operate. Because there are certain patterns with forces on different levels, the analysis distinguishes a first and a lower priority of mapping. However, for all the patterns it was possible to decide on which level the forces mainly operate.

To control for interrater effects, the second author performed an independent categorization for the first priority mapping of 20% of the patterns from each pattern collection (74 patterns). An interrater reliability analysis using the Kappa statistics was performed to determine consistency among raters. The interrater reliability was found to be high, with Kappa = 0.766 ($p < 0.000$), 95% CI (0.662, 0.870).

The analysis of the mappings has shown that indeed most of the patterns could be categorized into these layers (see 4.2 below) and a comparison of these mappings over all the collections shows interesting differences (see 4.3 below). There were also several patterns that could be mapped onto multiple layers (see 4.4 below). The detailed results of the analysis can be found on *http://goo.gl/OWQI2*

4.2 The analysis of the pattern collections

The first priority mapping of patterns from Tidwell (2006), van Duyne et al. (2007), van Welie (2009), and Yahoo! Inc. (2009) is shown in the following tables (Table 2 through Table 5).

4.2.1 Book "Designing Interfaces" by Jenifer Tidwell

The scope of Tidwell's (2006) patterns is the design of desktop applications, websites, web applications, and mobile devices. The patterns are rather platform-independent and they are illustrated with examples from several platforms.

Her book *Designing Interfaces* uses the following organization scheme for a total of 82 patterns:

- *Organizing the Content*
- *Getting Around*
- *Organizing the Page*
- *Doing Things*
- *Showing Complex Data*
- *Getting Input From Users*
- *Builders and Editors*
- *Making It Look Good*

All of these patterns could be classified into the layers of Baxley's model; there are no requirements and no software design patterns (see Table 2).

Table 2. Tidwell's categories (Tidwell, 2006) mapped onto the layers

	Tidwell's categories (number of patterns)								
Layer	*Organizing the Content*	*Getting Around*	*Organizing the Page*	*Doing Things*	*Showing Complex Data*	*Getting Input From Users*	*Builders and Editors*	*Making It Look Good*	**Total**
Requirements									**0**
Conceptual Model	1								**1**
Task Flow	5	5	1	2					**13**
Organizational Model									**0**
Viewing and Navigation	1	4	2		7				**14**
Editing and Manipulation			1			2	7		**10**
User Assistance	1	1	2	6	4	8	2		**24**
Layout			6	2	2				**10**
Style		1			1	1		7	**10**
Text									**0**
Software Design									**0**
Total	**8**	**11**	**12**	**10**	**14**	**11**	**9**	**7**	**82**

There are layers of Baxley's model that remain empty. There were no patterns for the *organizational model,* despite Tidwell discussing models of organization in the introduction to chapter 2. There were no patterns for *text* and only one for *conceptual model*. The other layers contain patterns, but most patterns are situated in the *behavior* tier. The *structure* tier contains the fewest patterns.

4.2.2 Book "The Design of Sites" by van Duyne et al.

The patterns of van Duyne et al. (2007) are written especially for the design of pre-Web 2.0 websites. The 107 patterns in their book, *The Design of Sites,* are organized in the following way:

- *Site Genres*
- *Creating a Navigation Framework*
- *Creating a Powerful Homepage*
- *Writing & Managing Content*
- *Building Trust & Credibility*
- *Basic E-Commerce*
- *Advanced E-Commerce*
- *Helping Customers Complete Tasks*
- *Designing Effective Page Layouts*
- *Making Site Search Fast & Relevant*
- *Making Navigation Easy*
- *Speeding Up Your Site*
- *The Mobile Web*

Most of these patterns could be fitted into Baxley's model (see Table 3) but 10 of them are software design patterns (e.g., *fast-loading images*) and 25 describe requirements patterns (e.g., *e-mail notifications*).

With the other patterns, all nine layers of Baxley's model are covered. The patterns are more or less evenly distributed across all the three tiers. It is interesting to see that there are some categories that fit directly into one tier of Baxley's model, whereas others show up in two tiers, and still others are spread across all three tiers; for example, the patterns under "site genres" all fit into the *conceptual model* layer, whereas the patterns from the chapter "the mobile web" are spread across all three tiers.

Table 3. Categories of van Duyne et al. (2007) mapped onto the layers

	Categories of van Duyne et al. (number of patterns)													
Layer	*Site Genres*	*Creating a Navigation Framework*	*Creating a Powerful Homepage*	*Writing & Managing Content*	*Building Trust & Credibility*	*Basic E-Commerce*	*Advanced E-Commerce*	*Helping Customers Complete Tasks*	*Designing Effective Page Layouts*	*Making Site Search Fast & Relevant*	*Making Navigation Easy*	*Speeding Up Your Site*	*The Mobile Web*	**Total**
Requirements		1		3	8	4	5	1			1	1	1	**25**
Conceptual Model	12													**12**
Task Flow		1	1			4	2	5			2			**15**
Organizational Model		6												**6**
Viewing and Navigation		1								2	6		1	**10**
Editing and Manipulation								2			1			**3**
User Assistance								4			3			**7**
Layout			1	2		1			6		1		1	**12**
Style					1						1			**2**
Text				3							2			**5**
Software Design				3				1		1		5		**10**
Total	**12**	**9**	**2**	**11**	**9**	**9**	**7**	**13**	**6**	**3**	**17**	**6**	**3**	**107**

4.2.3 Website "Welie.com" by Martijn van Welie

Earlier versions of van Welie's website distinguished the patterns in *web design patterns*, *GUI patterns*, and *mobile UI design patterns* but today, he just lists patterns in the categories below and the examples all seem to be from websites and web applications (van Welie, 2009).

The actual online catalogue *Welie.com* (May 2009) contains 131 patterns and has the following structure:

- User needs
    - *Navigating around*
    - *Basic interactions*
    - *Searching*
    - *Dealing with data*
    - *Personalizing*
    - *Shopping*
    - *Making choices*
    - *Giving input*
    - *Miscellaneous*
- Application needs
    - *Drawing attention*
    - *Feedback*
    - *Simplifying interaction*
- Context of design
    - *Site types*
    - *Experiences*
    - *Page types*

All but 18 of van Welie's patterns can be classified into Baxley's model. They are all requirements patterns and include all "experiences (context of design)" patterns and some of the "user needs" type patterns (e.g., the pattern *testimonials*). There are no software design patterns in this collection.

There are patterns distributed across most of the layers of Baxley's model but the *text* and the *organizational model* layers have no patterns. The other patterns are distributed more or less evenly across *structure* and *behavior* but with a few in the *presentation* tier. All but one of the "shopping" patterns fit into the *task flow* layer.

Table 4. Van Welie's categories (van Welie, 2009) mapped onto the layers

	Van Welie's categories (number of patterns)															
	User needs (83)									*Application needs (12)*			*Context of design (35)*			
Layer	*Navigating around*	*Basic interactions*	*Searching*	*Dealing with data*	*Personalizing*	*Shopping*	*Making choices*	*Giving input*	*Miscellaneous*	*Drawing attention*	*Feedback*	*Simplifying interaction*	*Site types*	*Experiences*	*Page types*	**Total**
Requirements						1	1	1	4	1				8	2	**18**
Conceptual Model													14			**14**
Task Flow		1	3	3	2	8	1	1							9	**28**
Organizational Model																**0**
Viewing and Navigation	25	4	5	8			2									**44**
Editing and Manipulation		1		2	1											**4**
User Assistance		1	4				1	1			2	2			2	**13**
Layout			1	1					1	3						**6**
Style										4						**4**
Text																**0**
Software Design																**0**
Total	**25**	**7**	**13**	**14**	**3**	**9**	**5**	**3**	**5**	**8**	**2**	**2**	**14**	**8**	**13**	**131**

4.2.4 Website "Yahoo! Design Pattern Library" by Yahoo! Inc.

The public patterns of the *Yahoo! Design Pattern Library* (Yahoo! Inc., 2009) cover different web design issues and many of them are Web 2.0 specific. The actual online catalogue contains 39 patterns (May 2009) and has the following structure:

- *Search*
- *Navigation*
- *Browsing*
- *Selection*
- *Rich interaction*
- *Social*

All but the "social" patterns, which can be seen mostly as requirements, could be classified into Baxley's model (see Table 5; Yahoo! categorized the pattern *search pagination* under "search" and "browsing", so by category the count of *viewing and navigation* patterns is 8 and the total count of patterns is 40). There are no software design patterns in Yahoo!'s pattern library.

Table 5. Yahoo!'s categories (Yahoo!, 2009) mapped onto the layers

	Yahoo!'s categories (number of patterns)						
Layer	*Search*	*Navigation*	*Browsing*	*Selection*	*Rich interaction*	*Social*	**Total**
Requirements						12	**12**
Conceptual Model							**0**
Task Flow						1	**1**
Organizational Model							**0**
Viewing and Navigation	(1)	4	2	1			**7**
Editing and Manipulation					1		**1**
User Assistance				2	15		**17**
Layout			1				**1**
Style							**0**
Text							**0**
Software Design							**0**
Total	**(1)**	**4**	**3**	**3**	**16**	**13**	**39**

In the Yahoo! library, only one pattern could be found that fitted into the *structure* tier (*sign-in continuity*) and the *presentation* tier (*page grids*), respectively; the other patterns all fitted into the *behavior* tier. What is interesting about Yahoo!'s organization of the patterns is that there are some higher-level patterns (e.g., *pagination*) that contain sub-patterns (*item pagination* and *search pagination* in this case) as different solutions to these higher-level patterns. Table 5 only counts the "leaf patterns"; i.e. the higher-level patterns (e.g., *pagination*) have not been counted.

4.3 The pattern collections in comparison

Table 6 shows an overview of the mappings of the different pattern collections onto our categories. It shows that some of the four collections cover all (van Duyne et al., 2007) or most of the nine layers (van Welie, 2009) of Baxley's model. Other collections focus on behavioral issues, with only a few patterns related to structure or presentation (Tidwell, 2006; Yahoo! Inc., 2009). Van Duyne et al.'s (2007) collection is the only one that also contains software design patterns (the others all focus on interaction design patterns) whereas Tidwell's (2006) is the only one that also contains no *requirements* patterns but is fully focused on the conceptual implementation of the user interface.

Table 6. Comparison of the four pattern collections (Tidwell, 2006; van Duyne et al., 2007; van Welie, 2009; Yahoo! Inc., 2009) by percentage (and by number of patterns)

Layer	*Tidwell*	*Van Duyne et al.*	*Van Welie*	*Yahoo!*	**Total**
Requirements		23.4% (25)	13.7% (18)	30.8% (12)	**15.3% (55)**
Conceptual Model	1.2% (1)	11.2% (12)	10.7% (14)		**7.5% (27)**
Task Flow	15.9% (13)	14.0% (15)	21.4% (28)	2.6% (1)	**15.9% (57)**
Organizational Model		5.6% (6)			**1.7% (6)**
Viewing and Navigation	17.1% (14)	9.3% (10)	33.6% (44)	17.9% (7)	**20.9% (75)**
Editing and Manipulation	12.2% (10)	2.8% (3)	3.1% (4)	2.6% (1)	**5.0% (18)**
User Assistance	29.3% (24)	6.5% (7)	9.9% (13)	43.6% (17)	**17.0% (61)**
Layout	12.2% (10)	11.2% (12)	4.6% (6)	2.6% (1)	**8.1% (29)**
Style	12.2% (10)	1.9% (2)	3.1% (4)		**4.5% (16)**
Text		4.7% (5)			**1.4% (5)**
Software Design		9.3% (10)			**2.8% (10)**
Total	**100% (82)**	**100% (107)**	**100% (131)**	**100% (39)**	**100% (359)**

In this overview, there are clearly identifiable gaps in the pattern collections. It is also apparent what other collections one could consider to fill the gaps. An overview of the original categories of the pattern collections shows that such a comparison is not possible there because the libraries only have categories where they have patterns and the different categories are very hard to reconcile (see Table 7).

Table 7. Original category names of the four pattern collections (Tidwell, 2006; van Duyne et al., 2007; van Welie, 2009; Yahoo! Inc., 2009)

Tidwell	**Van Duyne et al.**	**Van Welie**	**Yahoo!**
• *Organizing the Content*	• *Site Genres*	User needs:	• *Search*
• *Getting Around*	• *Creating a Navigation Framework*	• *Navigating around*	• *Navigation*
• *Organizing the Page*	• *Creating a Powerful Homepage*	• *Basic interactions*	• *Browsing*
• *Doing Things*	• *Writing & Managing Content*	• *Searching*	• *Selection*
• *Showing Complex Data*	• *Building Trust & Credibility*	• *Dealing with data*	• *Rich interaction*
• *Getting Input From Users*	• *Basic E-Commerce*	• *Personalizing*	• *Social*
• *Builders and Editors*	• *Advanced E-Commerce*	• *Shopping*	
• *Making It Look Good*	• *Helping Customers Complete Tasks*	• *Making choices*	
	• *Designing Effective Page Layouts*	• *Giving input*	
	• *Making Site Search Fast & Relevant*	• *Miscellaneous*	
	• *Making Navigation Easy*	Application needs:	
	• *Speeding Up Your Site*	• *Drawing attention*	
	• *The Mobile Web*	• *Feedback*	
		• *Simplifying interaction*	
		Context of design:	
		• *Site types*	
		• *Experiences*	
		• *Page types*	

Going back to the comparison using the unified categories (Table 6), one could see for example where to look to make Tidwell's (2006) collection more complete. It contains only one *conceptual model* and no *organizational model* patterns but the collection of van Duyne et al. (2007) has some of them. The missing *text* patterns could also be found there. Van Welie (2009) has more than three times the number of *viewing and navigation* patterns, so there might also be some potential here.

4.4 The patterns categorized into multiple layers

As mentioned above, there are several patterns that could be mapped onto multiple layers. This means that these patterns are not written with a clear focus in relation to these layers but describe aspects, which sometimes contain aspects of multiple layers in one pattern. An example from van Duyne et al. (2007) is the pattern "category pages", which covers aspects of *structure*, *behavior*, and *presentation*. The pattern describes how users should be able to navigate across different parts of a large site and that these parts should be organized into categories (*organizational model*). These categories should have a consistent navigation (*viewing and navigation*) placed in the layout in a consistent way (*layout*). The use of color coding (*style*) should support discrimination between these categories. Van Welie's (2009) pattern "home link" has aspects of the structure of navigational pathways (*task flow*) and of navigational elements (*viewing and navigation*).

5 Conclusions

We conclude with a discussion of the characteristics of the different interaction design approaches first. We then discuss Baxley's (2003) model and its extension, together with the analysis of the four pattern collections. In a final step, we present our concluding vision and suggest further steps that might be taken in order to achieve it.

5.1 The characteristics of different interaction design approaches

This study looks at several approaches to interaction design, to find out how the categorization of interaction design pattern languages can be enhanced by basing it on the structural characteristics of these approaches. Analysis of design approaches shows that most of them do not go into much detail in the stage where interaction design patterns come into play; i.e., when the user interface is designed. This raises an interesting question: why do most approaches see the task of designing the user interface rather like a black box? One explanation for this could be that experienced designers and researchers like Donald Schön (1984) consider the process of design as unpredictable to some extent. Schön sees design as an ongoing *reflective conversation* with the entity to be designed. Thus the path to the final design cannot be predicted; it is developed *de novo* in every project. This is why a linear process is not a realistic model for design activities. On the other hand, it can be argued that the performance of a *reflective practitioner* has certain distinguishable stages anyhow. Just as an artist would usually first use a pencil to sketch the overall form of a picture and later progress to oil-based paint to finish it, the work of the interaction designer moves through certain phases, building on each other. The different layered design processes can be seen as a task analysis in this sense. So even if the real process is not linear in nature, one can say that the organization of interaction design patterns, according to such a model, is "alongside the design process".

5.2 Baxley's model

We have identified Baxley's (2003) "Universal Model of the User Interface" as an interaction design approach with a very detailed description of the design phase. This model is based on common principles of design processes, such as a layered procedure, but its distinction of design sub-tasks is much more fine-grained than that of the other design processes. So it fulfills two important requirements of models for the organization of a pattern language: it is based on a design process and it provides enough "slots" to perform an effective organization. We have learned that the model does not cover all the interaction design patterns as defined by Dearden and Finlay (2006) but can be extended slightly to achieve this. However, 80% of all the analyzed patterns fall into the categories covered by Baxley's model. The model has been extended with a new category called "requirements patterns" as a working title: this is a category for the interaction design patterns that are "above" Baxley's model. For our analysis, this is detailed enough but for other pattern collections it may be valuable to sub-divide this category even further.

Even though Baxley's model has been developed in the context of web design, it seems to be robust enough to work on different platforms. The platforms can be seen as orthogonal to the model, and the "requirements patterns", as defined here, are ideally platform-independent. It can be said that the highest level of categorization with the tiers *structure*, *behavior*, and *presentation* is much more robust than the nine levels or even the sub-categories used by Baxley (2002). A voice user interface also has certain *structure*, *behavior*, and *presentation* aspects to it. But if one looks closer at the nine layers of Baxley's model, it can be seen that most notably the *presentation* tier is more focused on visual than on voice user interfaces. On the other hand, the speech of a voice user interface also has certain *"layout"* characteristics (e.g. order), a *"style"* (e.g. consistency to brand), and is made up of *"text"* (e.g. superfluous text should be avoided).

There are possible aspects to a user interface, such as audio and video, that are missing from the model. It would be necessary to add these categories to the model to make it more complete for other collections of patterns.

5.3 The analysis of the four pattern collections

To answer the question whether it is possible to organize interaction design patterns according to an extended version of Baxley's (2003) model, an analysis of four pattern collections was conducted. It cannot be guaranteed that this categorization is correct in every aspect, but we are convinced that this approach can make important aspects of pattern categorization visible. The aim of this work is *not* to discuss how individual patterns are categorized but to show the benefits of taking such a "unified" top-down approach to pattern categorization for the construction of pattern languages.

One result of the analysis is that all of the patterns focusing on interaction design could be fitted into these categories. The analysis of the four pattern collections has also shown that some of them cover most or all of the categories. However, some collections focus on behavioral issues with only few patterns related to structure or presentation issues. From this, it can be concluded that some collections are more complete than others, in the sense that they span the whole problem space of user interface design in a systematic and detailed way. The extended model provides a stable framework of “problem slots”.

These conclusions indicate that the use of such a model helps to build a pattern language that is not only *complete*, but also actively supports the process of identifying new patterns; i.e., it is *predictive*. In striving for a complete pattern language, one is forced to describe the full range of “user interface problems” and has to find all the common solutions to a given level of the model. For the individual patterns, the implication is that they cannot only describe solutions on a merely morphological or behavioral level (i.e. how they look or work) but must also describe them on a semantic level (i.e. what they are used for in the user interface). A certain user interface mechanism can be used for different purposes; for example the component of a drop-down list box can be used for filtering a table (*viewing and navigation*) or for the selection of different given values for data entry (*editing and manipulation*). Following this thinking, one is forced to describe not the user interface mechanisms per se but the solution to the user’s problems.

5.4 A common model for the organization of interaction design patterns

As Gabriel (1996) reports from the area of software engineering, patterns are not used enough in projects in this field and we think that the same can be said for the field of HCI. From our point of view, one of the biggest problems is that existing pattern collections are far from complete and it is not easy for the individual designer to know where to look for missing patterns. This searching for patterns is time-consuming. The vision of this work is for pattern collections to be written on the same basis – for example, based on the extended model presented here – enabling individual patterns to move freely between collections to make them more complete.

Even with the extended model as defined above, there are still some problems to be solved before the vision can be fulfilled. There are patterns that do not fit unambiguously into the model; they have mixed forces according to the levels of the model. So if one were to write patterns with forces precisely formulated in relation to the model, the particular patterns would therefore be accurately separated; this would lead to clearer identification of the levels on which gaps in a certain pattern collection exist.

Another issue is that of platform. As it has been shown above, there are patterns that are platform-independent and others that are not. We think that it makes no sense to demand patterns to be fully platform-independent in every case. This would unreasonably limit the range of issues that can be described as interaction design patterns. However, it would be a requirement to describe individual patterns as platform-independent as possible to make them available on a wide range of technical platforms. Then, in order to know which patterns can be moved from one collection to another, the patterns need to be tagged by platform-dependence to filter them accordingly.

5.5 Outlook

This work shows why Baxley's model may be a good foundation for the organization of interaction design patterns to achieve pattern collections that allow a free transfer of patterns. But there are several other milestones to be met before this vision can be achieved. The first is that although the logic of the model seems to work there are several gaps in it (audio, video, etc.) that have to be filled to make it universally applicable. To handle large numbers of patterns, the category of "requirements patterns" on the one hand and the nine layers of Baxley's model on the other should be made more fine-grained in a robust way. Also, new trends in the technology have to be taken into account. It is not the aim of this work to prove that Baxley's model is the only one capable of performing such a task, but rather to show the advantages of such a "unified model". For the moment, no better model could be found.

Because in this study we have focused more on the *construction* of complete interaction design pattern languages, in the next stage it would be important to focus on the *usage* of these languages by interaction designers. It would be very interesting to investigate whether interaction designers engaged in a design activity could specify – when interrupted and asked – to which level of the model this activity belonged.

There is another issue that must be mentioned: This work focuses on the structural aspects of pattern organization. As mentioned in the introduction, there are two main aspects to pattern organization: the organization according to several *levels of design* and the *network* of patterns defined by reference. There is already some research about the linking of design patterns (van Welie and van der Veer, 2003), but it would be interesting to explore this question to a further extent, also taking into consideration the findings presented here.

Acknowledgements

The authors would like to thank Zürcher Kantonalbank (ZKB) in Zurich, Switzerland for the support and funding of this research as part of the UCD ZKBconnect project.

References

Alexander, C. (1964). *Notes on the synthesis of form*. Harvard University Press, Cambridge.

Alexander, C. (1979). *The timeless way of building*. Oxford University Press, New York.

Alexander, C. (1999). The origins of pattern theory: the future of the theory, and the generation of a living world. *IEEE software*, 16(5):71–82.

Alexander, C., Ishikawa, S., and Silverstein, M. (1977). *A pattern language: towns, buildings, construction*. Oxford University Press, New York.

Baxley, B. (2002). *Making the Web work: defining effective Web applications*. New Riders.

Baxley, B. (2003). Universal model of a user interface. In *Proceedings of the 2003 conference on Designing for user experiences*, pages 1–14. ACM New York, NY, USA.

Beyer, H. and Holtzblatt, K. (1998). *Contextual design: defining customer-centered systems*. Morgan Kaufmann, San Francisco, CA.

Borchers, J. (2000). Interaction design patterns: Twelve theses. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, volume 2, page 3.

Borchers, J. (2001). *A pattern approach to interaction design*. Wiley, Chichester, England.

Constantine, L. L. and Lockwood, L. A. D. (1999). *Software for use: a practical guide to the models and methods of usage-centered design*. Addison Wesley, Reading, Mass.

Cooper, A., Reimann, R. and Cronin, D. (2007). *About face 3: the essentials of interaction design*. Wiley Pub., Indianapolis, IN.

Dearden, A. and Finlay, J. (2006). Pattern languages in HCI: A critical review. *Human-Computer Interaction*, 21(1):49–102.

Dimitriadis, Y., Goodyear, P., and Retalis, S. (2009). Using e-learning design patterns to augment learners' experiences. *Computers in Human Behavior*, 25(5): 997–998.

Fincher, S. and Windsor, P. (2000). Why patterns are not enough: some suggestions concerning an organising principle for patterns of UI design. In *CHI'2000 Workshop on Pattern Languages for Interaction Design: Building Momentum*.

Fincher, S. (2002). Patterns for HCI and Cognitive Dimensions: two halves of the same story. In *Kuljis, J., Baldwin, L., Scoble, R., Proceedings of the Fourteenth Annual Workshop of the Psychology of Programming Interest Group*, pages 156–172.

Fincher, S. (2003). PLML: Pattern language markup language report of workshop held at CHI September 2003. *Interfaces*, 56 (pp. 26–28).

Gabriel, R. (1996). *Patterns of software: tales from the software community*. Oxford University Press New York.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co. Inc., Boston.

Garrett, J. J. (2002). *The elements of user experience: user-centered design for the web*. New Riders, Indianapolis, Ind.

Granlund, Å., Lafrenière, D., and Carr, D. A. (2001). A pattern-supported approach to the user interface design process. In *Proceedings of 9th International Conference on Human-Computer Interaction HCI International*.

Grudin, J. (1992). Utility and usability: research issues and development contexts. *Interacting with computers*, 4(2): pages 209–217.

Hornbaek, K. (2006). Current practice in measuring usability: Challenges to usability studies and research. *International Journal of Human-Computer Studies*, 64(2): pages 79–102.

IBM Corporation. (1992). *Object-Oriented Interface Design: IBM Common User Access Guidelines*. QUE, New York.

Kohls, C., Uttecht, J. (2009). Lessons learnt in mining and writing design patterns for educational interactive graphics. *Computers in Human Behavior* 25 (5), 1040–1055.

Leacock, M., Malone, E., and Wheeler, C. (2005). Implementing a pattern library in the real world: A Yahoo! case study. In *American Society for Information Science and Technology Information Architecture Summit*, Montréal, Québec, Canada.

Leuthold, S., Schmutz, P., Bargas-Avila, J.A., Tuch, A.N. & Opwis, K. (2011). Vertical versus dynamic menus on the world wide web: Eye tracking study measuring the influence of menu design and task complexity on user performance and subjective preference. *Computers in Human Behavior*, 27(1), 459-472.

Mayhew, D. J. (1999). *The usability engineering lifecycle: a practitioner's handbook for user interface design*. Morgan Kaufmann Publishers, San Francisco, Calif.

Nielsen, J. (1993). *Usability engineering*. Academic Press, Boston.

Pauwels, S.L., Hübscher, C., Leuthold, S., Bargas-Avila, J.A., & Opwis, K. (2009). Error prevention in online forms: Use color instead of asterisks to mark required fields. *Interacting with Computers,* 21(4), 257-262.

Pauwels, S., Hübscher, C., Bargas-Avila, J., and Opwis, K. (2010). Building an interaction design pattern language: A case study. *Computers in Human Behavior,* 26(3), pages 452-463.

Rantzer, M. (1996). *Field methods casebook for software design*: The delta method – a way to introduce usability, pages 91–112. John Wiley & Sons, Inc., New York, NY.

Robert, D. (1998). *Designing for the user with OVID: bridging user interface design and software engineering*. Software engineering series. Macmillan Technical Pub., Indianapolis, IN.

Schön, D. (1984). *The Reflective Practitioner: How Professionals Think In Action*. Basic Books.

Sharp, H., Rogers, Y., and Preece, J. (2007). *Interaction Design*. Wiley, New York, 2nd edition.

Tidwell, J. (2006). *Designing interfaces*. O'Reilly, Beijing.

Tidwell, J. (2009). *Common Ground: A Pattern Language for Human-Computer Interface Design*. Retrieved May 7, 2009, from http://www.mit.edu/~jtidwell/common_ground.html

van Duyne, D. K., Landay, J. A., and Hong, J. I. (2002). *The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience*. Addison-Wesley Professional.

van Duyne, D. K., Landay, J. A., and Hong, J. I. (2007). *The design of sites: patterns for creating winning web sites*. Prentice Hall, Upper Saddle River, NJ, 2nd edition.

van Welie, M. and Trætteberg, H. (2000). Interaction patterns in user interfaces. In *Proc. Seventh Pattern Languages of Programs Conference: PLoP*, pages 13–16.

van Welie, M. and van der Veer, G. (2003). Pattern Languages in Interaction Design: Structure and Organization. In *Proceedings of Interact*, volume 3, pages 1–5.

van Welie, M. (2009). *Patterns in interaction design*. Retrieved May 7, 2009, from http://www.welie.com/patterns/

Yahoo! Inc. (2009). *Design pattern library*. Retrieved May 29, 2009, from http://developer.yahoo.com/ypatterns/

Analyzing the gap between the results of foundational user research and the final design of a user interface

Christian Hübscher, Mirjam Seckler,
Alexandre N. Tuch, Klaus Opwis

*University of Basel, Department of Psychology,
Center for Cognitive Psychology and Methodology, 4055 Basel, Switzerland*

Abstract

This work explores how, in the area of designing user interfaces, the gap between problem domain and solution domain can be bridged. It is very important to better understand this translation between these two domains because interaction designers have to do it again and again in their daily work. We first analyze the anatomy of these two domains and explain the nature of the gap. In this context, interaction design patterns can be seen as "building blocks" to construct the resulting user interface. Therefore, the organization of patterns can be taken as a description of the solution domain. In the discussion of the different means to bridge the gap between these domains two such ways are highlighted. The *bridging of the distance* between the results of foundational user research and the final design of the user interface can be achieved mainly with different phases, iterations, and intermediate representations. Alternatively, bridging the gap between problem domain and solution domain as a whole can be described as a *mapping* between the different parts of these two domains. As not much work about such a mapping has yet been undertaken, it is analyzed here in detail. Finally, these different means of bridging the gap are brought together and the possible impact of these findings are discussed.

Keywords: user-centered design; interaction design; design patterns; pattern languages

1 Introduction

User-centered design (UCD) in the area of human-computer interaction (HCI) means the following: Foundational user research is undertaken to understand the needs of the users and all other relevant aspects of the context of usage. Based on this knowledge, the user interface is designed. The resulting designs are tested with users and refined iteratively. *Interaction design* in the area of UCD means that these user research results have to be translated into a concept for a user interface. This translation is at the very heart of the interaction designer's task. Interaction designers also need great knowledge of generic best practices in *user interface design*. Much work has already been carried out in the areas of methods for foundational user research and best practices for user interface design (see, e.g., Hackos and Redish, 1998; Courage and Baxter, 2004; see also, Mayhew, 1992; Cooper et al., 2007; Shneiderman and Plaisant, 2004). However, there is very *little* material available that describes in detail the *translation* from results of foundational user research into a user interface. However, this translation is what interaction designers have to do in their projects *again and again*. Therefore it is important that HCI practitioners not only have proven methods of user research and generic interface design advice but also have support in achieving this translation. Wood (1997b) has already mentioned this gap between user requirements and the design of the user interface.

The work presented here shows a way of closing this gap even further. First, the problem and the solution domain for interaction design will be analyzed. The *problem domain* comprises aspects that build all the preconditions for interaction design. That is what is analyzed in foundational user research. The *solution domain* is the "design space" in which the user interface has to be designed. These two domains are shown to be very different in their internal topology. Following on from these differences, the *gap* between the two domains will be highlighted. The gap can be described as either a *distance to be bridged* or the need for a *mapping* between the two domains. Regarding the aspect of bridging the distance, a lot of work has already been done, mainly with the breakdown into different phases, with iterations, and with intermediate representations; however, there is not much material to be found for a mapping between the two domains. Therefore we present a mapping of the different aspects of the problem domain to the different levels of the solution domain. Finally, we show a combination of these different means to bridge the gap.

2 Problem domain, solution domain, and the gap between the two

Interaction design can also be described in the following way: In order to design a user interface, the designer has to choose from a body of several "building blocks" and combine them. In order to know which blocks to use, the designer has to understand *the needs of the* *users* and all the other *aspects of the context of usage*. The interaction designer has to know the *problem domain* very well. The *solution domain* consists of the range of all possible user interfaces that could be designed. Another part of the solution domain are the laws about what types of “building blocks” are needed in order to design a complete user interface and the rules about how such blocks can be combined. Therefore, in the context of the solution domain we will also discuss *interaction design patterns* as one possible form that these “building blocks” could have.

Because the two domains contain very different sets of information, it can be said that there is a certain gap between them. In this part of our work, we want to show the anatomy of these two domains and explain the nature of the gap.

2.1 Problem domain

The problem domain consists of all aspects of the world that have to be taken into consideration for designing a user interface. In order to come up with an exhaustive description of the problem domain, we have analyzed several UCD processes to see what they consider to be relevant. We have looked into the work of Nielsen (1993), Rantzer (1996), Beyer and Holtzblatt (1998), Roberts et al. (1998), Mayhew (1999), Constantine and Lockwood (1999), Garrett (2002), Baxley (2002), and Cooper et al. (2007) to find out how these authors describe the problem domain. All aspects mentioned in these different works are listed in Table 1.

Because the problem domain is described in very distinct ways in the works mentioned above, we have looked for a way to categorize the individual aspects listed by the authors. Shackel (2009) has come up with a model that is well established (see Day et al., 2009). This model says that there are “four principal components of any user-system situation: user, task, tool and environment.” (Shackel, 2009, p. 339). We took these four components – *user*, *environment, task* and *tool* (which will be explained in more detail later) – to categorize the contents of Table 1. Most of the aspects could be categorized into one of the four components. However, a few items did *not* fit into any of these four categories but they could be grouped into a new category, called *project*. This category covers certain facets of a typical UCD project *itself*. These facets include project objectives, interests of different stakeholders, project resources, etc. and are mentioned by Garrett (2002), Baxley (2002), and Cooper et al. (2007).

Table 1. Analysis of UCD lifecycles concerning problem domain tasks

UCD lifecycle	***Aspects of the problem domain***				
	User	**Environment**	**Task**	**Tool**	**Project**
	The four categories taken from Shackel (2009)				*A new category*
Nielsen (Nielsen, 1993)	Individual user characteristics The evolution of the user and the job		The user's current and desired tasks Functional analysis		
Delta Method (Rantzer, 1996)	User profiling		Task analysis		
Contextual Design (Beyer & Holtzblatt, 1998)		Flow model Cultural model Physical model Artifact model	Sequence model		
OVID (Roberts et al., 1998)	User classes User goals		Current tasks User objects		
Usability Engineering Lifecycle (Mayhew, 1999)	User profile		Task analysis	Platform capabilities and constraints General design principles	
Usage Centered Design (Constantine & Lockwood, 1999)	Role model	Operational model	Task model		
The Elements of User Experience (Garrett, 2002)	User needs				Site objectives
Universal Model of a User Interface (Baxley, 2002)	User needs			Technical constraints	Competition Business opportunity Core design values Opportunity statement
Goal Directed Design (Cooper et al., 2007)	User interviews & observations: understand user *needs* Personas: user & customer archetypes	Other models: represent domain factors beyond individual users & customers	User interviews & observations: understand user *behavior*		Scope: define project goals & schedule Audit: review existing work & product Stakeholder interviews: understand product vision & constraints

The results of this analysis have been taken to draw the map shown in Figure 1. Here, only the structure of the problem domain and the influences within this domain are explained. The influences on the solution domain are explained in section 4.

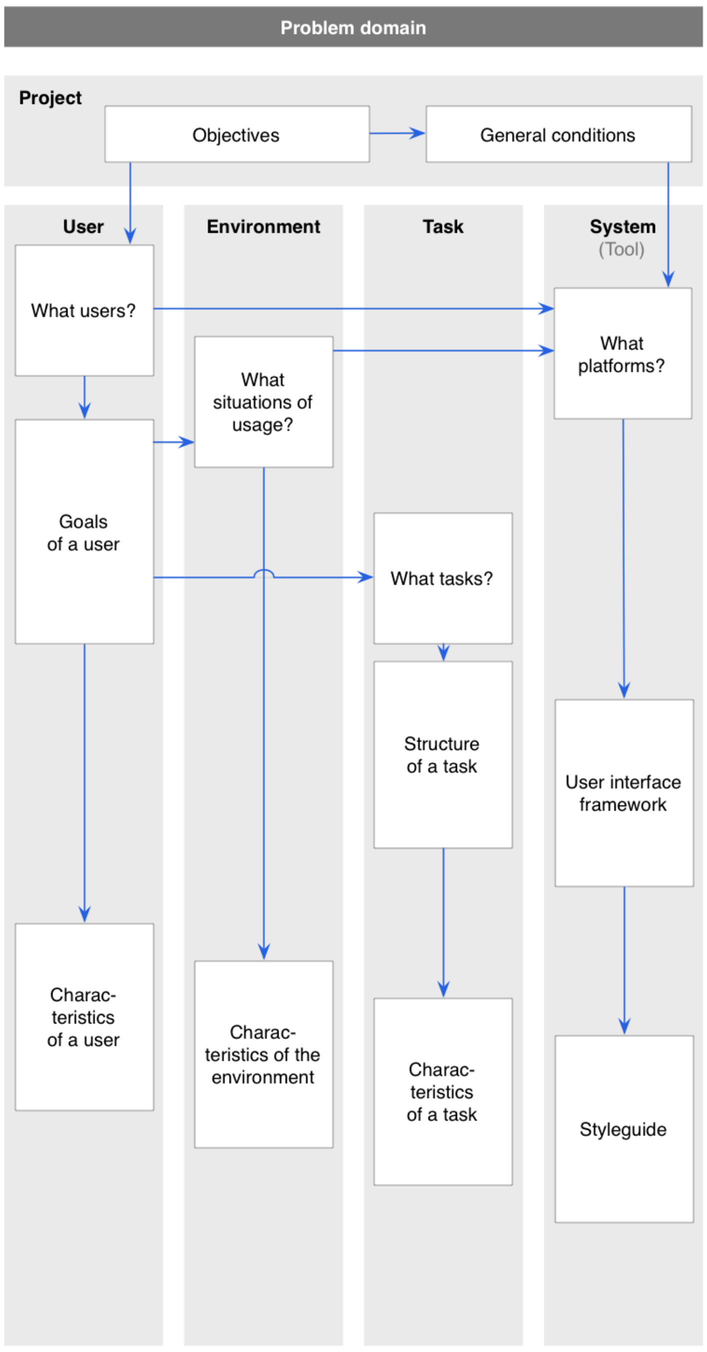


Figure 1. The problem domain in detail

2.1.1 Project

This can be seen as the setup of the whole undertaking. The basic components of a *project* setup are the "objectives" and the "general conditions". Most UCD approaches remain silent about these project setup tasks, but recent approaches such as Garrett (2002), Baxley (2002), and Cooper et al. (2007) stress the importance of these first steps. Based on the "objectives" of a project, it should be clear "what users" are the important ones. The "general conditions" can be seen as preconditions for the *system*. For example, often some technical aspects are already fixed and cannot be challenged. They have an influence on the *system* (tool) part of the problem domain. The different aspects of a *project* can also have a direct impact on the solution domain.

2.1.2 User

The totality of all the *users* can be segmented into user groups or modeled with personas (Cooper, 1999). This step is about the different types of users and is called "what users". In a UCD approach, the question "what platforms" should partly be answered based on the prioritization of the user groups. The individual user groups will then be further analyzed in more detail. The most important aspects of a group of users are the "goals of a user" (as stressed by Cooper, 1999). The goals influence "what situations of usage" are probable; they also influence "what tasks" have to be supported. Together with the goals, there are also "characteristics of a user" that are relevant (e.g., domain knowledge, age). All these aspects can influence the solution domain, at least indirectly.

2.1.3 Environment

In the *environment*, in which the users use a system, different aspects influence the fitness of a solution. The question "What situations of usage?" influences "what platforms" have to be supported. For example, if a system is used while commuting with public transport, mobile platforms have to be supported. There are also "characteristics of the environment" that have to be taken into account such as lighting, noise. All these aspects can also have a direct influence on the solution domain.

2.1.4 Task

In the analysis of the *tasks* there are several things to consider. First, an analysis is carried out of "what tasks" are to be supported. For every task, further analysis is needed. In regards to the "structure of a task", one wants to know what sub-tasks and individual steps there are. But the "characteristics of a task" are also relevant such as importance and frequency of the individual tasks. These aspects can all directly influence the solution domain.

2.1.5 System (Tool)

According to Shackel (2009), a user-system situation consists of user, task, tool, and environment. In this sense, *tool* is the implemented solution with the final user interface design. Therefore most of it is part of the solution domain, but there are also aspects of the tool that are given as preconditions for the interaction design. These preconditions are part of the problem domain and are henceforth called *system*.

The first question of the *system* component is "what platforms" have to be supported. Depending on the technical platforms the corresponding "user interface framework" is given. In most cases these frameworks define the supported interaction techniques and given standard user interface components. On most technical platforms these frameworks are implemented as a set of APIs (application programming interfaces), which allow the developers to use some modules without having to program them themselves (see e.g., "Cocoa" in Apple, 2016a). Together with the user interface frameworks most platforms also have their "styleguide" (also called *guidelines* in many cases, see e.g., Apple, 2016b; SAP, 2016), which gives guidance on the visual aspects and texts in a user interface. For the sake of simplicity in this work, the aspects "user interface framework" and "styleguide" are clearly separated. Here the "user interface framework" contains *structure* and *behavior* facets of the user interface and the "styleguide" *presentation* facets (see section 2.2 for more about *structure*, *behavior*, and *presentation*). In reality, this distinction is not possible in this way and how these aspects are addressed depends on the platform. Existing *interaction design patterns* of a certain platform can also be seen as part of the framework or the styleguide. In the context of the problem domain, they are the set of *all possible* patterns that can be chosen from, in order to design a solution (see section 2.2.1 for the role of patterns in the context of this work). These different *system* aspects can all directly influence the solution domain.

2.2 Solution domain

The solution domain can be described as the assembling of several interaction design patterns until the user interface has been designed in all of its dimensions. If one takes an organizational scheme for such patterns, this provides a "map" of the solution domain (see Figure 2). Therefore the role of interaction design patterns and their organization is explained in more detail below.

The anatomy of the solution domain, as it is presented here, is based on our recent research into the organization of interaction design patterns (see Hübscher et al., 2011). It is based on the logic of Baxley's model (see Baxley, 2002, 2003), which is similar to that of Garrett (2002). The logic of these approaches is that their individual layers have to be based on the preceding layers (see Baxley, 2002; Garrett, 2002). As these two authors emphasize, this should *not* to be mistaken for a waterfall approach, but has to be done iteratively.

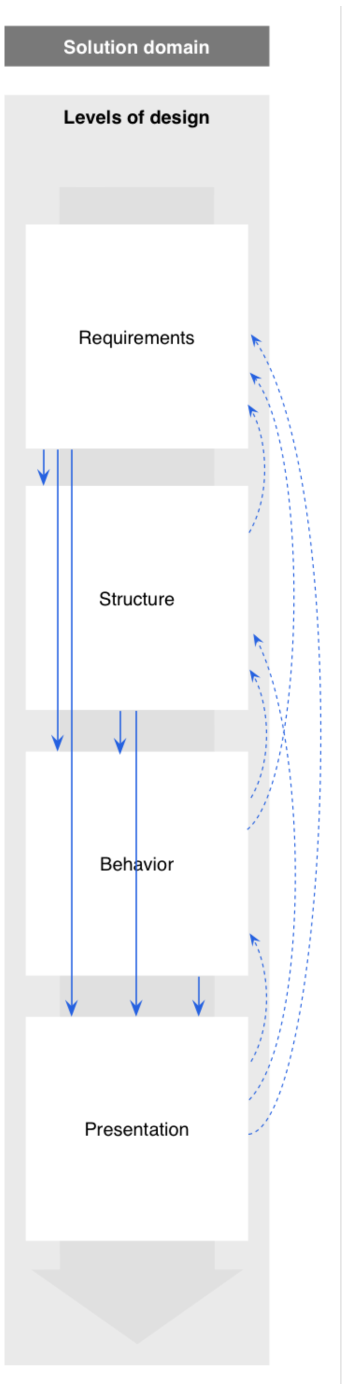


Figure 2. Illustration of the solution domain

2.2.1 The role of interaction design patterns

This work is based on the assumption that a user interface can be composed entirely of interaction design patterns. The patterns are on different levels (see Figure 3 for examples). In this way the whole user interface can be assembled with these “building blocks”. This is also how Baxley (2002) describes the process of interaction design in his approach. However, most pattern libraries are not designed in a way that this is possible, because most of them have “holes” on certain levels (see Hübscher et al., 2011).

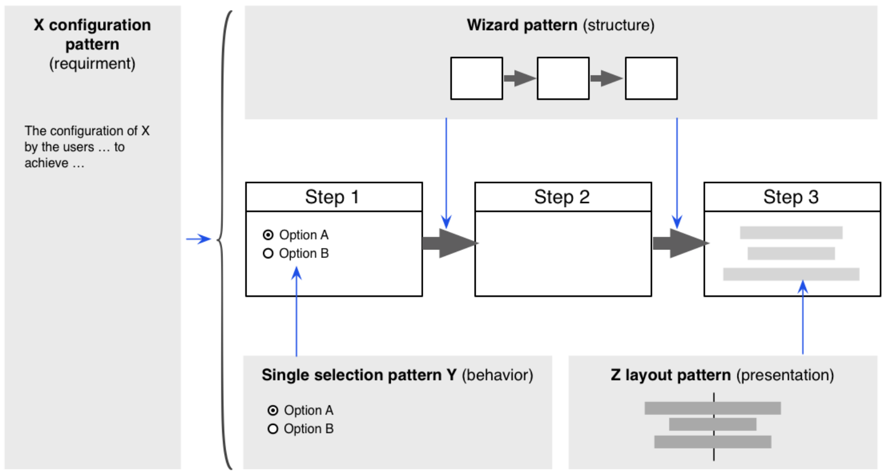

X configuration pattern (requirment)
The configuration of X by the users … to achieve …
Wizard pattern (structure)
Step 1
Option A
Option B
Step 2
Step 3
Single selection pattern Y (behavior)
Option A
Option B
Z layout pattern (presentation)

Figure 3. Patterns on different levels of the solution domain

In the context of this work, the patterns can be formally described as those coming from a pattern library (such as Tidwell, 2006; van Duyne et al., 2007; Yahoo! Inc., 2009; van Welie, 2009) or from the experience of a seasoned designer.

2.2.2 The organization of interaction design patterns

A coherent collection of patterns is called a *pattern language* (Dearden und Finlay, 2006). The organizational scheme for a pattern language can be seen as a “map” for the solution domain, because a complete pattern language should cover its whole solution domain (see Dearden und Finlay, 2006; van Welie and van der Veer, 2003). Such an organizational scheme has been presented in Hübscher et al. (2011). It is based on the nine layers of Baxley’s (2003) model and is extended with a new category called “requirements patterns” (see Figure 4). Thus, this gives it even more similarities with the approach of Garrett (2002).

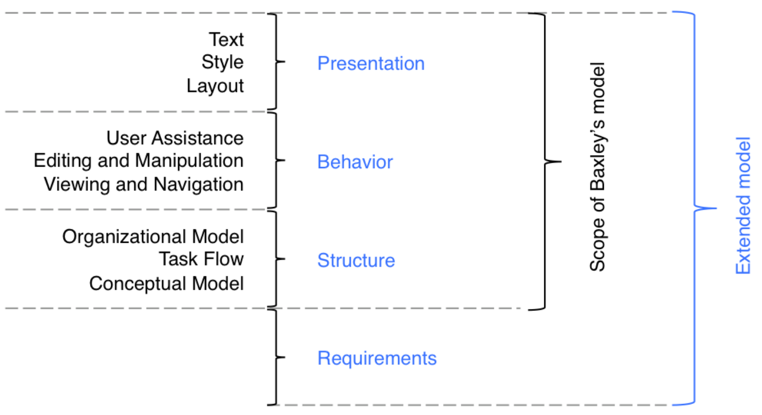


Figure 4. The extended model for the organization of interaction design patterns (adapted from Hübscher et al., 2011, based on Baxley, 2003)

This extended model is used in the work here to subdivide the solution domain. In the following, the four main levels *requirements*, *structure*, *behavior*, and *presentation* are described in more detail. More examples explaining the levels can be found in Hübscher et al. (2011) and Baxley (2002).

2.2.3 The requirements level

The most basic level of the solution domain is that of *requirements*. In the context of this work, the relevant requirements are those that have a perceivable impact on the user interface. There are two types of such requirements (see Wiegers and Beatty, 2013 for the different ways to discern requirements). The *non-functional* requirements say *how* a user interface is designed, in a general, overall sense. The *functional* requirements say *what* the tasks are that are supported; that is, what functionality should be implemented. A certain requirement can influence the structure of a solution but it can also have a direct impact on its behavior or presentation. The functional requirement "e-mail notifications" (van Duyne et al., 2007) could be implemented as a simple user interface element with a pattern on the level of *behavior* or if the feature is more complex it could also influence the *structure* of the user interface. The non-functional requirement "site accessibility" (van Duyne et al., 2007) can contain aspects with an influence on the level of *behavior*, such as the use of invisible navigation mechanisms to support users with screen readers. Or it can influence the *presentation* level with the need for heightened contrast to support visually impaired users.

2.2.4 The structure level

The next level is the *structure* of a user interface. This level is about patterns such as conceptual models, page types, workflows, and information architectures. Baxley (2003) defines the enclosing levels *conceptual model*, *task flow*, and *organizational model* here. These patterns are not so much about the contents of individual screens but more about the organization of several screens together. On this level it is decided whether, for example, a *wizard pattern* or a *guide pattern* (see Baxley, 2002) should be used. The structure of a user interface can further influence its *behavior* and *presentation*.

2.2.5 The behavior level

The patterns on the level of *behavior* are about the direct interaction of the user with the user interface. Baxley (2003) names the enclosing levels of *viewing and navigation*, *editing and manipulation*, and *user assistance*. Those include mechanisms for navigation, searching, filtering, data entry, help, alerts, etc. The patterns on the level of behavior can then use *presentation* patterns to support them. For example, certain user interface elements, such as important buttons (*behavior*), can be supported in their visibility with a certain color for emphasis (*presentation*).

2.2.6 The presentation level

The patterns on the *presentation* level concern the design of the visual appearance (color, icon style, etc.) or the writing of texts (alerts, menu items, etc.) in a user interface. Baxley (2003) describes the enclosing levels of *layout*, *style*, and *text*. They could also concern other audio-visual aspects of a user interface not mentioned by Baxley (2002).

2.3 The gap between problem domain and solution domain

After the analysis of the problem and the solution domain, we focus on the *gap* between the two. Wood (1997b) describes the gap as follows:

> … while there are some excellent sources of information on user interface design, none contains specific descriptions on how a designer *transforms* the information gathered about users and their work into an effective user interface design. … Some might argue that is to be expected because that process is a highly creative one and that creative processes are inexplicable by their nature. While this may be true in a limited sense, designs don't really appear as if by magic. [italics in the original] (Wood, 1997, p. 3)

In the book edited by Wood (1997a), different authors describe this gap in various ways (Graefe, 1997; Ludolph, 1997; Scholtz and Salvador, 1997; Simpson, 1997). Dirbach et al. (2011) describe the field of software engineering in a similar way. This material has inspired much of our approach, focusing on two separate aspects.

There is a first aspect of the gap that is mentioned by many authors (see section 3.1), even if they describe it slightly differently. Here, it is called the *distance to be bridged* between foundational user research and final design. This distance can be seen as the changing levels of abstraction of the different steps' results between foundational user research and the final user interface design.

A second, more unusual way of looking at the gap is to see it as a result of the different structures of the problem and the solution domain. From this point of view, the bridging can be seen as a *mapping* of the individual parts of the two domains. When one looks at the illustrations of the problem domain in section 2.1 and the solution domain in section 2.2 it is striking that their setup is very different. As an example, results from foundational user research provide neither the ideal navigation structure nor the ideal font size for a user interface. Rather, one learns something about users and their tasks. From these user research results, designers then have to derive the design of a user interface.

3 Existing research about the bridging of the gap

There is already some research about the bridging of the gap; however, much more material is available on the aforementioned first way of looking at the gap than on the second.

3.1 Distance to be bridged

In relation to the final design of a user interface, the results of foundational user research have a *level of abstraction* that is very high. In order to "overcome this distance" there are different ways to segment it into sections. The three main ways to do this are:

- *Different phases*
- *Iterations*
- *Intermediate representations*

All authors in Wood's (1997a) book use some methods of segmenting their approach into *different phases*. Authors of different UCD approaches also do this (including those mentioned above: Nielsen, 1993; Rantzer, 1996; Beyer and Holtzblatt, 1998; Roberts et al., 1998; Mayhew, 1999; Constantine and Lockwood, 1999; Garrett, 2002; Baxley, 2002; Cooper et al., 2007). Possible examples of design phases are *conceptual design* and *prototyping* (Rantzer, 1996). More examples can be found in Hübscher et al. (2011). Such phases also give advice on the sequence in which intermediate representations should be worked out. In the work here, to a great extent, this aspect is already contained implicitly in the definitions of problem and solution domain (see sections 2.1 and 2.2.).

Iteration is another important approach to breaking down the distance between analysis and design. Iterative approaches have a long history in computer science (see Larman and Basili, 2003). But they are also an established mandatory aspect to successful design in UCD (see International Organization for Standardization, 2010; Dow et al., 2009). As the use of different phases, iteration is a concept that is mentioned throughout Wood's (1997a) book and it is also an important part of the different UCD approaches mentioned above.

A lot of material exists in the area of *intermediate representations*. The main groups of intermediate representations are models, sketches and prototypes. The *models* can represent aspects of the problem domain *or* the solution domain. They can be informal or formal, abstract or concrete. Models in the solution domain are a way to visualize certain aspects of the user interface before doing so with the more concrete sketches or prototypes.

Some established *models in the problem domain* are:

- *Personas* (Cooper, 1999; Goodwin, 2009)
- *User roles* and *task cases* (Constantine and Lockwood, 1999)
- *Scenarios* (Carroll, 2000)
- The five *work models* of Beyer and Holtzblatt (1998)
- *User object models* (Van Harmelen, 2001)
- *Task organizational models* (Mayhew, 1999)

Examples of *models in the solution domain* are:

- *Navigation maps* and *abstract prototypes* (Constantine and Lockwood, 1999)
- *User environment design* (Beyer and Holtzblatt, 1998)
- *Storyboards* (Greenberg et al., 2012)
- *Wireframes* (Brown, 2007)
- *Mood boards* (Endrissat et al., 2016)

Prototypes are the most concrete and maybe the representation with the most material. Research has been carried out into the different techniques of prototyping (e.g., Warfel, 2009; Snyder, 2003; Arnowitz et al., 2007). There is a plethora of prototyping tools on the market. More theoretical treatises on prototyping can be found in Houde and Hill (1997), Lim et al.

(2008), and McCurdy et al. (2006). See Buxton (2007) for the difference between sketches and prototypes.

3.2 Mapping between problem and solution domain

Another way of bridging the gap between the two domains is with the description of the various influences of aspects of the problem domain on different levels of the solution domain. This approach is mentioned less often in existing research discussing the gap. Nevertheless, a number of authors provide some insight into how to map aspects of the problem domain on the solution domain (e.g., Graefe, 1997; Ludolph, 1997; Scholtz and Salvador, 1997; Simpson, 1997). However, these descriptions mostly describe mappings between certain intermediary results, rather than between the problem and the solution domain as a whole. If more direct mappings are mentioned, they focus on individual aspects only. To the authors' knowledge no overview exists of the different mappings between these two domains.

There are many individual guidelines about single mappings between aspects of the problem domain and certain levels of the solution domain. Work on *design principles* (e.g., Cooper et al. 2007; Mayhew, 1992; U.S. Dept. of Health and Human Services, 2006; Matrai, 2010; Johnson, 2007) as well as collections of *interaction design patterns* (Tidwell, 2006; van Duyne et al., 2007; van Welie, 2009; Yahoo! Inc., 2009) provide valuable sources. Based on such material, we will analyze the mapping of the problem domain on the solution domain in the next section.

4 The mapping

Analysis of the mapping is outlined in this section, organized by the four levels of the solution domain: requirements, structure, behavior, and presentation. As a "unit" of such a mapping in the solution domain, we assume having several interaction design patterns. These patterns can be formally described or those from other sources. It is not the aim of this work to list all the possible patterns but rather to describe exemplary patterns in order to make the various mappings clear. There are aspects of the problem domain that have a direct influence on the solution domain and there are some that do this indirectly in combination with other aspects of the problem domain. The effects *within* the problem domain have already been described in section 2.1.

4.1 Requirements level

Figure 5 shows that part of the mapping affecting the level of requirements. The “units” of this mapping are these interaction design patterns, which we call *requirements patterns* (see section 2.2.3 above). In Table 2 there are examples for requirements patterns in relation to the different aspects of the problem domain.

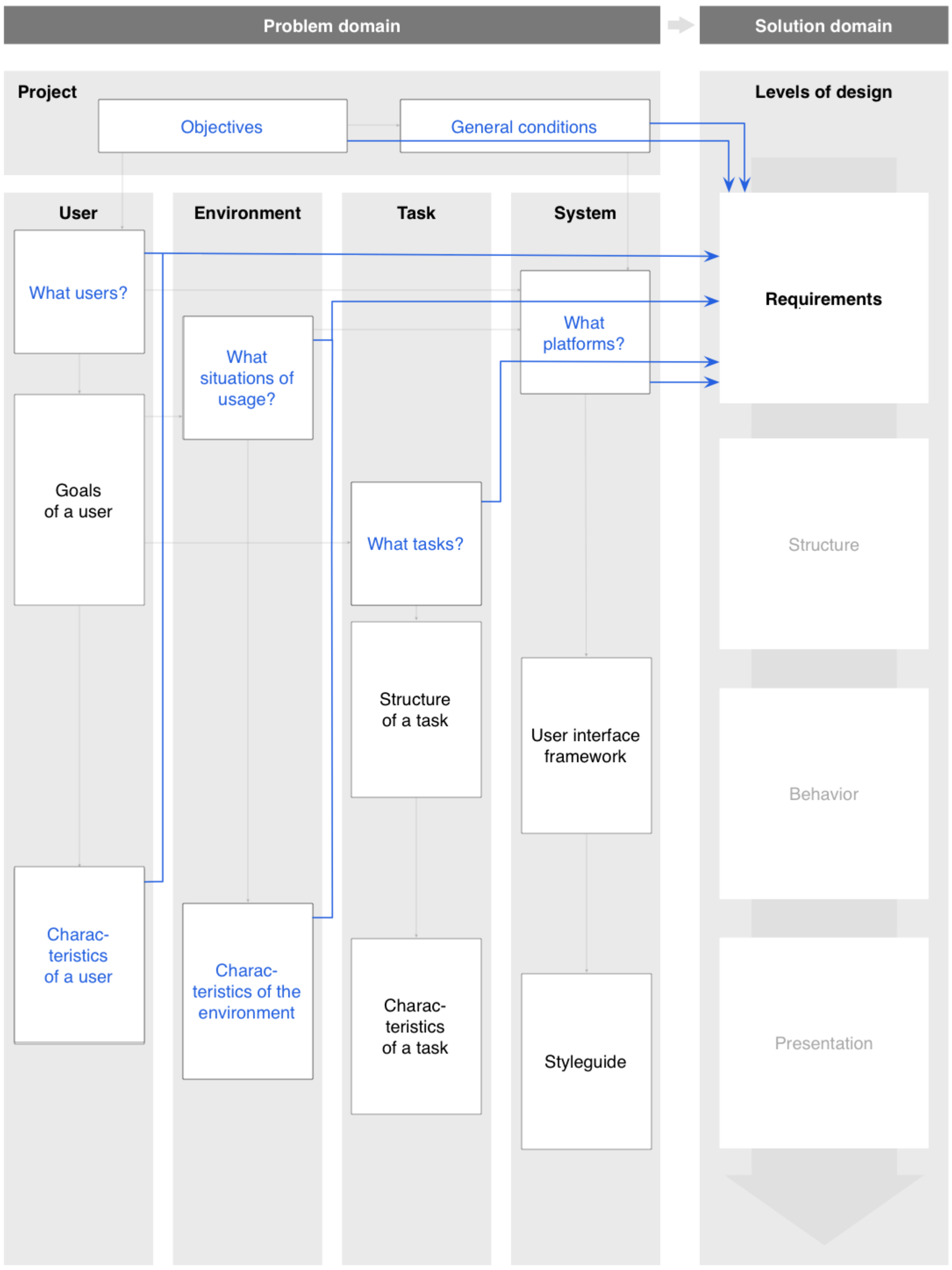


Figure 5. Mapping on the level of requirements

At the design level, *requirements'* most essential task is to define the different types of user interfaces that are needed. This depends on the number of primary personas (see Cooper et al., 2007; Goodwin, 2009), but also on the platforms to support. General requirements concerning the design of the user interface depend on the characteristics of the users such as accessibility issues (e.g., Pfeil et al., 2009), the different situation of usage (e.g., day and nighttime usage), and individual characteristics of these situations (e.g., a noisy environment). Furthermore, there exist requirements arising from the objectives and general conditions of a project.

Table 2. Requirements patterns

Problem domain	***Type of pattern***	***Examples***	***References***
Objectives	Essential functions (derived from main objectives of a project)	Applications on different platforms to share, sync, and store files across devices	Eisenmann et al. (2012)
General conditions	General conditions for the design of the user interface	Limitations of hardware available influence what is possible in the user interface	Perkins et al. (1997)
What users?	Different user interfaces for different primary personas	User interfaces for expert users and for beginners	Cooper et al. (2007); Goodwin (2009)
Characteristics of a user	User interfaces for different types of users	Accessible user interfaces; user interfaces for seniors	Fink et al. (1997); Peissner (2011); Pfeil et al. (2009); Karahasanovića (2009)
What situation of usage?	User interfaces for *different or changing* contexts	User interfaces for day and nighttime use	Schmidt et al. (1999); Dey (2001)
Characteristics of the environment	*Optimizing* a user interface for a *certain* context	A user interface for a noisy environment	Gerfelder et al. (2000); Kunc et al. (2013)
	Functions for support of context	Magnification of an interface area to facilitate clicking	Mankoff et al. (2000)
What tasks?	Main functions	A mapping of tasks to functions	Maguire & Bevan (2002)
What platforms?	User interfaces for different platforms	A web version and a mobile version of the user interface	Heller & Rivers (1996); Eisenstein et al. (2000)

4.2 Structure level

Figure 6 shows the graphical mapping from the problem domain on the level *structure* of the solution domain. See Table 3 for examples of patterns on the *structure* level.

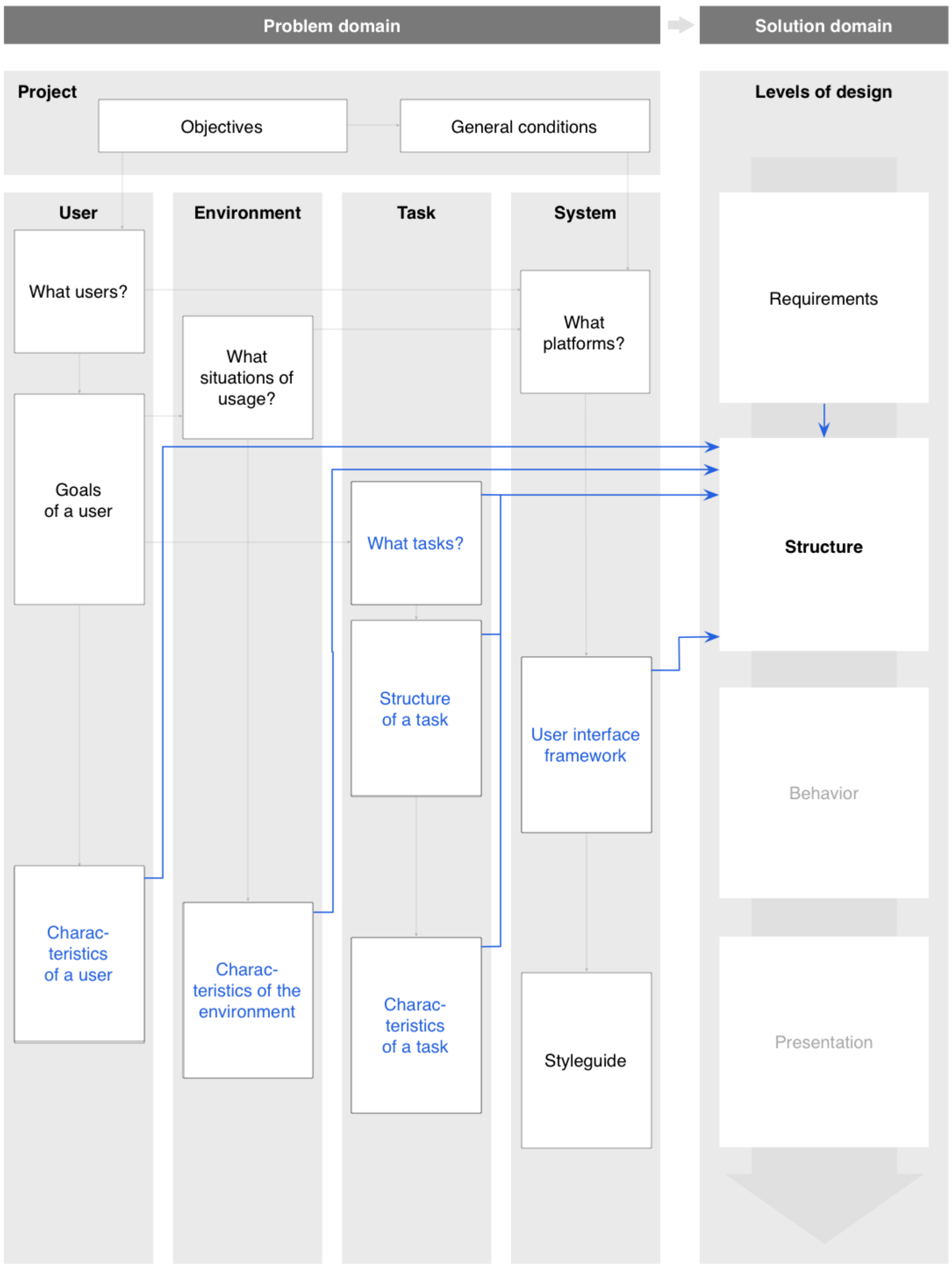


Figure 6. Mapping on the level of structure

To define the structure of a user interface, one has to consider several aspects of the problem domain: It is essential to know the characteristics of a user in order to match the conceptual model of the interface (e.g., Roth et al., 2010; Zhang, 2008) and to organize the functions based on the user's mental model (e.g., Young, 2008). Furthermore, all aspects concerning *task* of the problem domain are crucial for the structure of an interface. Research about what task a user wants to perform influences the conceptual model (e.g., Yadav, 2010) and the organization of functionality based on the number and the types of tasks to support (e.g., Vu and Proctor, 2011) also have an influence on how the structure of the interface should be defined. Moreover, the structure of the task (e.g., process navigation following a certain business process, see Johnson et al., 2000) as well as its overall structure (e.g., Kosters et al., 1996) should be considered. Characteristics of a task furthermore help to decide the placement of important functions on different views of a user interface (Tidwell, 2006), and characteristics of the environment help design the solution's structure to support the structures found in the environment (e.g., a process navigation following the structure of paper documents used by users, Fouse et al., 2011). Finally, the user interface framework has to be considered to define the types of processes or interaction contexts provided by the platform used (e.g., Apple, 2016b; Microsoft, 2016).

Table 3. Patterns concerning structure

Problem Domain	***Type of Pattern***	***Examples***	***References***
Characteristics of a user	Conceptual model based on the mental model of the user	Metaphors such as shopping cart, album	Roth et al. (2010); Zhang (2008)
	Organization of functions based on mental model of the user	Topical navigation (based on card-sorting results)	Young (2008)
Characteristics of the environment	User interface structure following the structures found in the environment	Process navigation following the structure of paper documents used by users	Fouse et al. (2011)
What tasks?	Conceptual model based on the tasks	Conceptual model for information-retrieval task	Yadav (2010)
	Organization of functionality based on the number and the types of tasks to support	Topical navigation (based on hierarchical task analysis)	Vu & Proctor (2011)
Structure of a task	User interface structure following the structure of the task	Process navigation following a certain business process	Johnson et al., (2000)
	Organization of functionality based on the overall structure of the main task	Task-based organizational scheme	Kosters et al., (1996)
Characteristics of a task	Placement of important functions on different views of a user interface	Placing an "escape hatch" on each page, which brings the user back to a known place	Tidwell (2006)
User interface framework	Types of processes or interaction contexts provided by the platform used	Assistant (Mac OS) or wizard (Windows); Window; Panel; Pane	Apple (2016b); Microsoft (2016)

4.3 Behavior level

Figure 7 shows the mapping from the problem domain to the *behavior* level and Table 4 shows examples of such patterns.

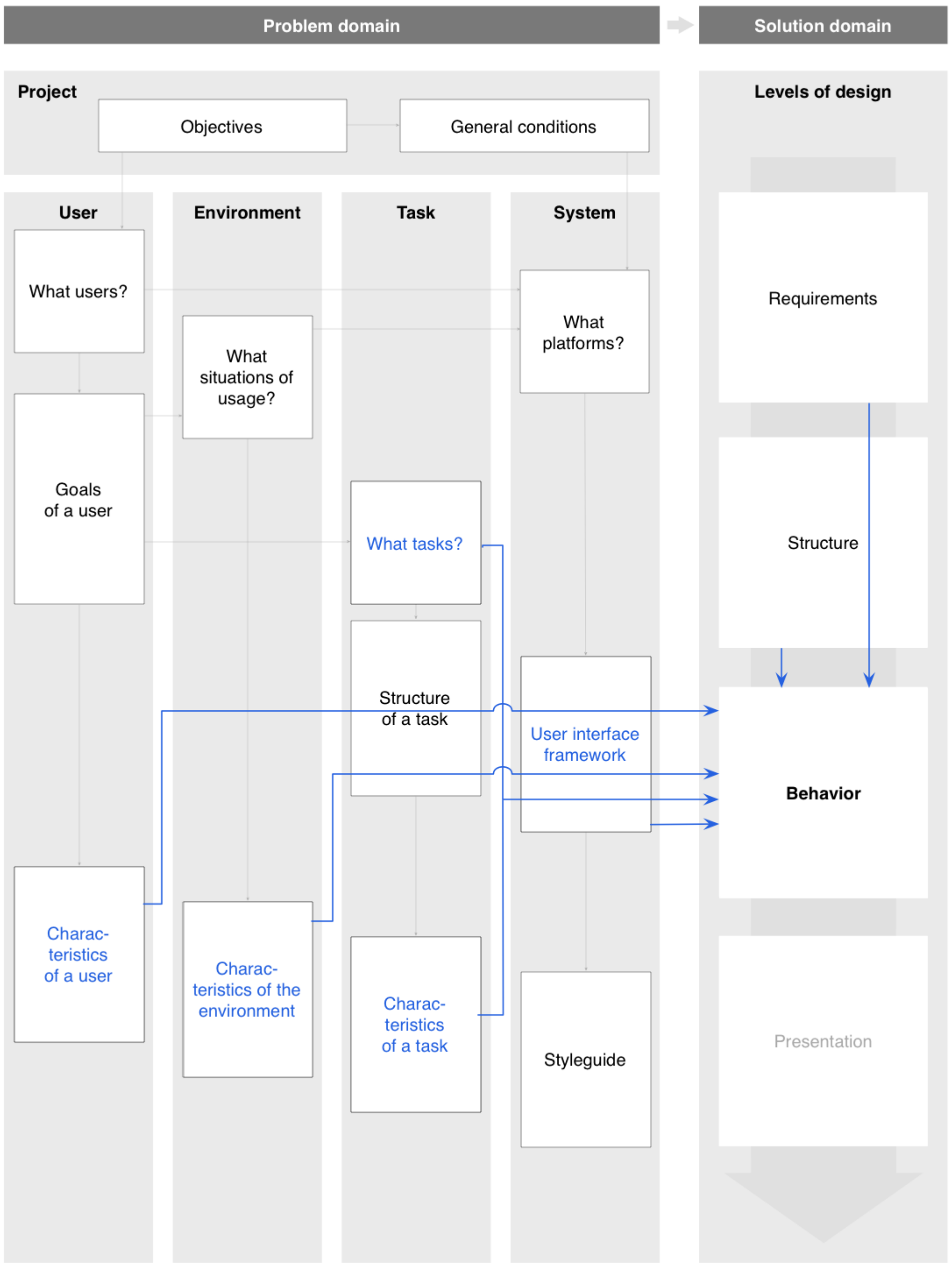


Figure 7. Mapping on the level of behavior

Aspects from the problem domain encountered at the behavior level include the *characteristics of a user, characteristics of the environment, what tasks, characteristics of a task,* as well as the *user interface framework.* Typical types of patterns involve mechanisms and assistance of an interface. For aspects regarding users, mechanisms and assistance help to support users' abilities (Findlater and McGrenere, 2004) or special user groups (Lunn and Harper, 2011). Characteristics of the environment for example include mechanisms that are best for a certain situation of usage (e.g., tangible interactions vs. graphical interfaces, see Horn et al., 2012). Aspects regarding tasks include mechanisms that are best for a certain user goal (Cutrell and Guan, 2007) and support different variants of tasks (Linderman and Fried, 2004). The user interface framework, finally, could show different mechanisms available on a platform that influence the behavior level for example.

Table 4. Patterns concerning behavior

Problem Domain	***Type of Pattern***	***Examples***	***References***
Characteristics of a user	Mechanisms best for users abilities	Dynamic vs. static menus	Findlater & Mc Grenere (2004)
	Assistance best for certain users	Providing assistance to older users of dynamic web content	Lunn & Harper (2011)
Characteristics of the environment	Mechanisms best for a certain situation of usage	Situations in which tangible interaction seems to offer advantages over graphical interfaces for learning	Horn et al. (2012)
What tasks?	Mechanisms best for a certain user goal	Search vs. browsing	Cutrell & Guan (2007)
Characteristics of a task	Mechanisms best for different variants of a tasks	Textboxes vs. radio buttons, checkboxes or drop-down menus in web forms	Linderman & Fried (2004)
	Assistance best for certain tasks	Adaptive user interfaces that support routine situations	Lavie & Meyer (2010)
User interface framework	Mechanisms available on a platform	A system that considers a user's position in a task when reasoning about when to interrupt	Bailey et al. (2006)

4.4 Presentation level

Figure 8 highlights the mappings from the problem domain to the *presentation* level and Table 5 shows examples of such patterns.

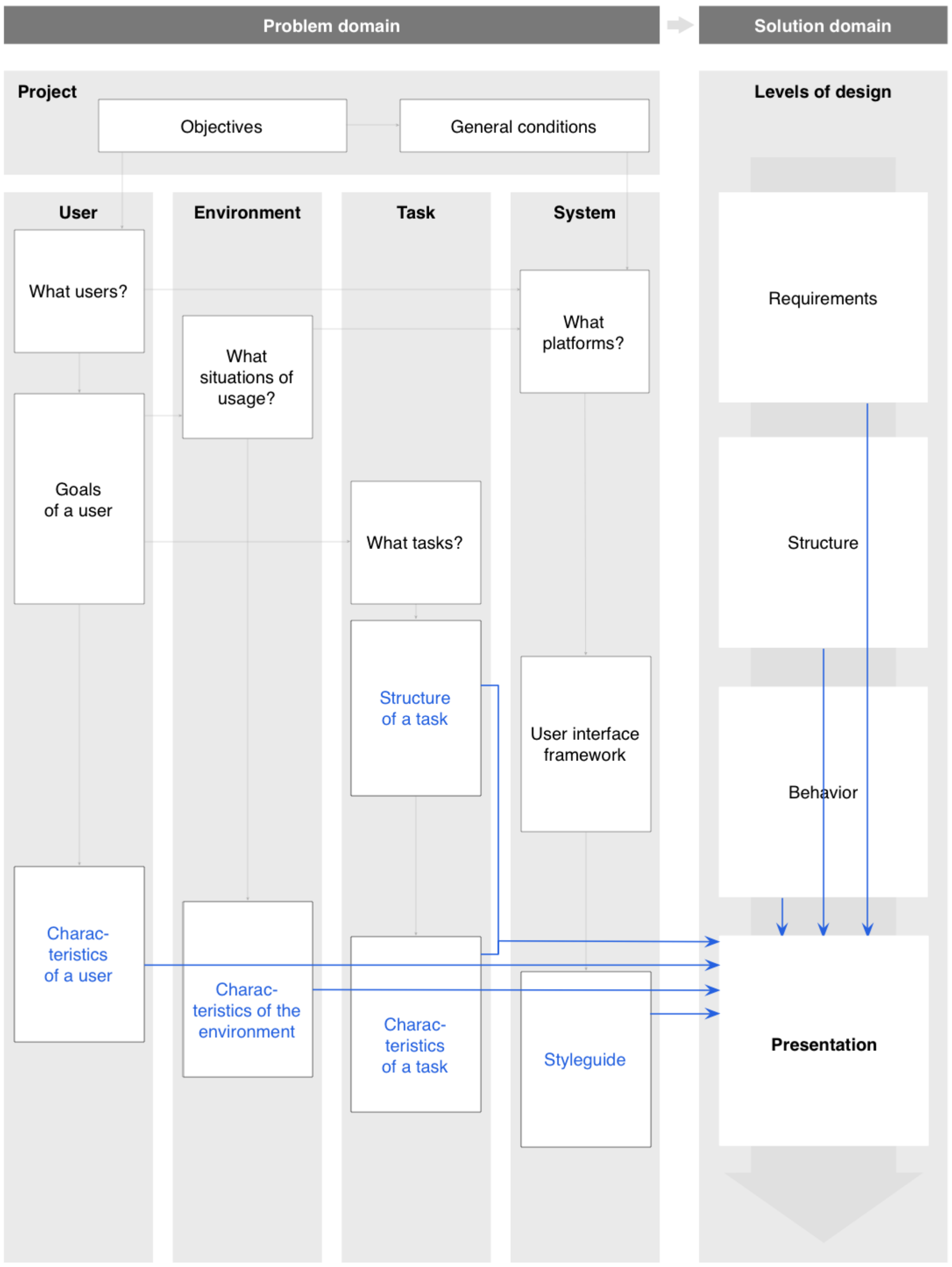


Figure 8. Mapping on the level of presentation

Linderman and Fried (2004) highlight the importance of using a text style that is appropriate to the user (for example, for error messages). George et al. (2011) showed the importance of the layout to fit the user and Taslim et al. (2009) revealed that the style is important to suit the user, as children, for example, differ from adults in their preferences for interface type, font type, and background color. Patterns about the characteristics of the environment concern the text and the style (Hall et al., 2002; Schleicher et al., 2011). The layout should also help to support the structure of a task (Chang and Nesbitt, 2006; Thalheim and Düsterhöft, 2000) and characteristics of a task (Gruber, 1995; Microsoft, 2016; Wu and Yuan, 2003). Finally, the system layer in the problem domain is represented by the styleguide and should consider standard text, styles and layouts (Becker and Mottay, 2001; Kascak et al. 2013).

Table 5. Patterns concerning presentation

Problem Domain	***Type of Pattern***	***Examples***	***References***
Characteristics of a user	Text style to fit the user	Simple and understandable error messages	Linderman & Fried (2004)
	Layouts to fit users	Layout for right to left reading users	George et al. (2011)
	Style that suits user	Colorful appearance for kids	Taslim et al. (2009)
Characteristics of the environment	Match text to text used in environment of usage	Labels similar to those in some of the supporting tools used by the users	Schleicher et al. (2011)
	Style to fit the environment	High contrast for low light conditions	Hall et al. (2002)
Structure of a task	Layouts to fit structure of a task	Structuring of a dialog to fit the workflow	Thalheim & Düsterhöft (2000)
	Make structures visible	Gestalt principles to make structures more visible and consistent	Chang & Nesbitt (2006)
Characteristics of a task	Placement of functions in the layout	Important task placed prominently in the layout	Microsoft (2016)
	Make some functions visible	Highlighting of dangerous functions by red color	Wu & Yuan (2003)
	Match text to task domain	Use of task specific labels (e.g., for designing ontologies)	Gruber (1995)
Styleguide	Standard layouts	Same layout for different languages	Becker & Mottay (2001)
	Standard styles	Icon styles	Kascak et al. (2013)
	Standard text	Standard button labels	Becker & Mottay (2001)

5 Combination of the different ways to bridge the gap

After analyzing the mapping between problem and solution domain we will explore what a combination of *distance bridging* and *mapping* could mean. Then, possible influences on the creation of *interaction design pattern languages* are shown together with *design rooms* as a possible area of the results' direct application.

5.1 A combination of mapping and distance bridging

Figure 9 shows a combination of mapping between the two domains and a bridging of the distance. As mentioned in section 3.1, the main ways to bridge the distance between problem and solution domain are the use of *different phases*, *iterations*, and *intermediate representations*. Here, in this analysis, the focus is on intermediate representations. They are aligned alongside of several phases and some of them are mapped according to the different design levels. The choice of variants of models, prototypes, and other methods are not meant to be exhaustive; their only purpose is to illustrate the ideas presented here.

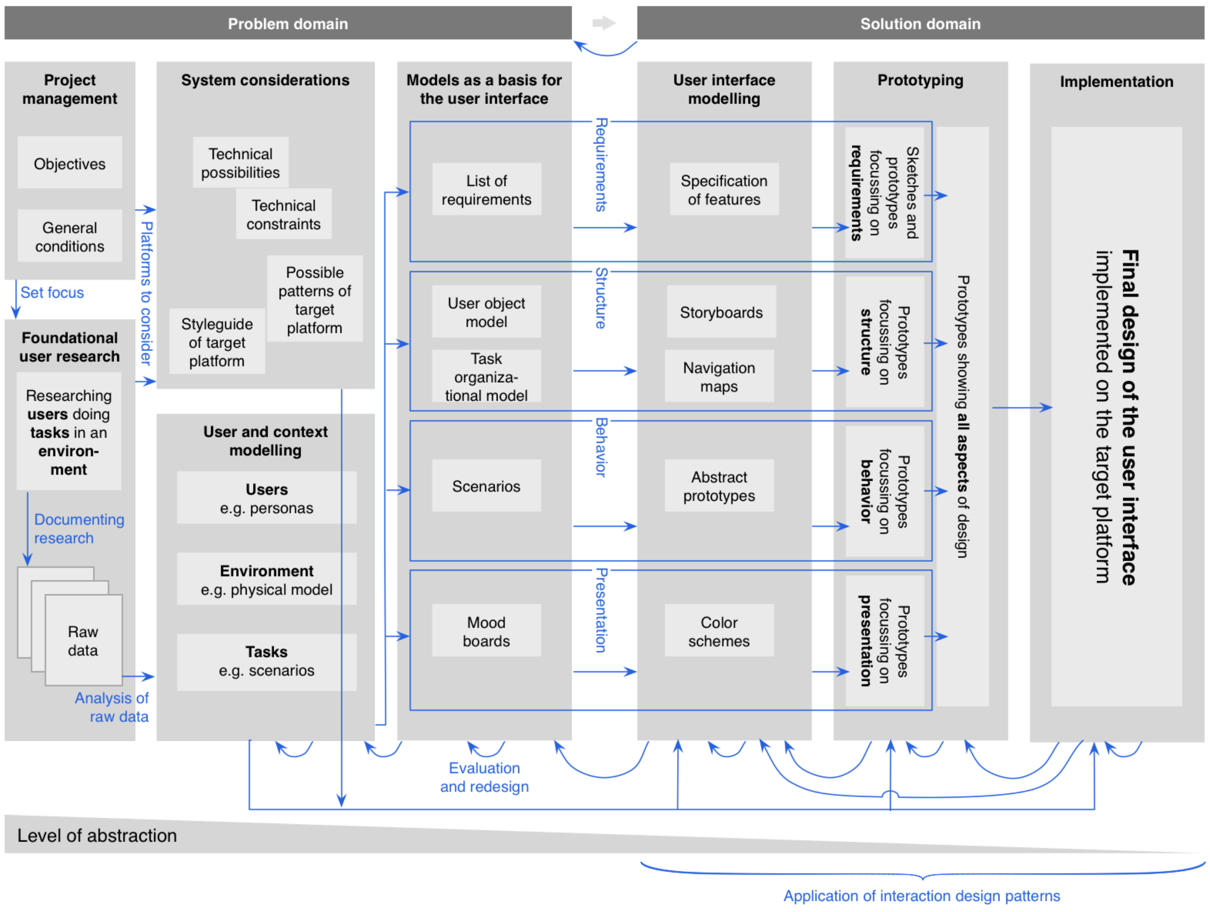

Problem domain
Solution domain
Project management
Objectives
General conditions
Set focus
Foundational user research
Researching users doing tasks in an environment
Documenting research
Raw data
Analysis of raw data
Platforms to consider
System considerations
Technical possibilities
Technical constraints
Possible patterns of target platform
Styleguide of target platform
User and context modelling
Users e.g. personas
Environment e.g. physical model
Tasks e.g. scenarios
Models as a basis for the user interface
List of requirements
User object model
Task organizational model
Scenarios
Mood boards
Requirements
Structure
Behavior
Presentation
User interface modelling
Specification of features
Storyboards
Navigation maps
Abstract prototypes
Color schemes
Prototyping
Sketches and prototypes focussing on requirements
Prototypes focussing on structure
Prototypes focussing on behavior
Prototypes focussing on presentation
Prototypes showing all aspects of design
Implementation
Final design of the user interface implemented on the target platform
Evaluation and redesign
Level of abstraction
Application of interaction design patterns

Figure 9. Combination of mapping and distance to be bridged

The *objectives and general conditions of the project* as well as the *results of foundational user research* have the highest level of abstraction in relation to the final design. The *raw data* of user research is condensed into models for the *users*, the *environment,* and the *tasks*. Examples of such models are: *personas* (Cooper, 1999) for the users, the *work models* of Beyer and Holtzblatt (1998) for the environment, and *scenarios* (Carroll, 2000) for the tasks. These models are still very distant to the final design, but they are stripped of information not relevant for interaction design. If the technical platform is given, its *technical possibilities* and *constraints* as well as the *possible interaction design patterns* and *styleguide* have to be taken as a basis for interaction design.

The next phase contains the *models*, which can be seen as a direct *basis for the user interface*. They are still part of the problem domain, but because they are mapped on the different levels of the solution domain, they are much closer to the final design than the set of previously mentioned models. On the level of *requirements*, the example is not a model per se but a *list* *of requirements* that result in certain *features*. A *user object model* (Van Harmelen, 2001) and a *task organizational model* (Mayhew, 1999) can be seen as the foundation for the *structure* of the user interface. In the solution domain, this structure can be modeled with *storyboards* (Greenberg et al., 2012) and *navigation maps* (Constantine and Lockwood, 1999). On the level of *behavior*, there can be detailed *scenarios* (Carroll, 2000) to describe details of the various tasks. Functions for these tasks can be visualized with models such as *abstract prototypes* (Constantine and Lookwood, 1999). Certain aspects relevant for *presentation* can be captured in *mood boards* (Endrissat et al., 2016), which can directly inform, for example, the construction of a *color scheme* (see chapter "Color" in Google, 2016, as an example).

Further concretization within the solution domain is the use of *prototypes*. A first step can be to focus prototypes on the different levels of design. A prototype can show the different *requirements*, without taking into account other aspects of interaction design. These prototypes can be called *role prototypes* (Houde and Hill, 1997). Buxton (2007) calls such representations, which are meant to explore *what* to build, *sketches*. Other prototypes can focus on the different levels of *structure*, *behavior*, or *presentation*. These kinds of prototypes can be called *look and feel prototypes* (Houde and Hill, 1997). Even closer to the final solution are *prototypes, which show all aspects of design together* in a balanced way. If they consider all the mentioned aspects together with technical feasibility, they can be called *integration prototypes* (Houde and Hill, 1997). Based on all these models and prototypes, the final design is implemented on the target platform.

The concepts of *iteration* and *evaluation* are included in Figure 9, showing that models and prototypes should be iterated together with their evaluation and redesign.

5.2 Design room

One possible direct application of these results could be in the setup of a design room. A *design room* is a room that is permanently available for a design team to work in, but also a place to put the important preliminary project results on its walls. This helps the team always to have the important information in view and not lose sight of the big picture. Several authors advocate the use of such rooms in interaction design (Karat and Bennett, 1991; Rantzer, 1997; Rohlfs, 1997; Simpson, 1997; Beyer and Holtzblatt, 1998). The grid shown in Figure 9 can be taken as a grid for a wall in a design room in order to have an overview of the transformation of foundational user research results into a user interface (see Figure 10 for an example).

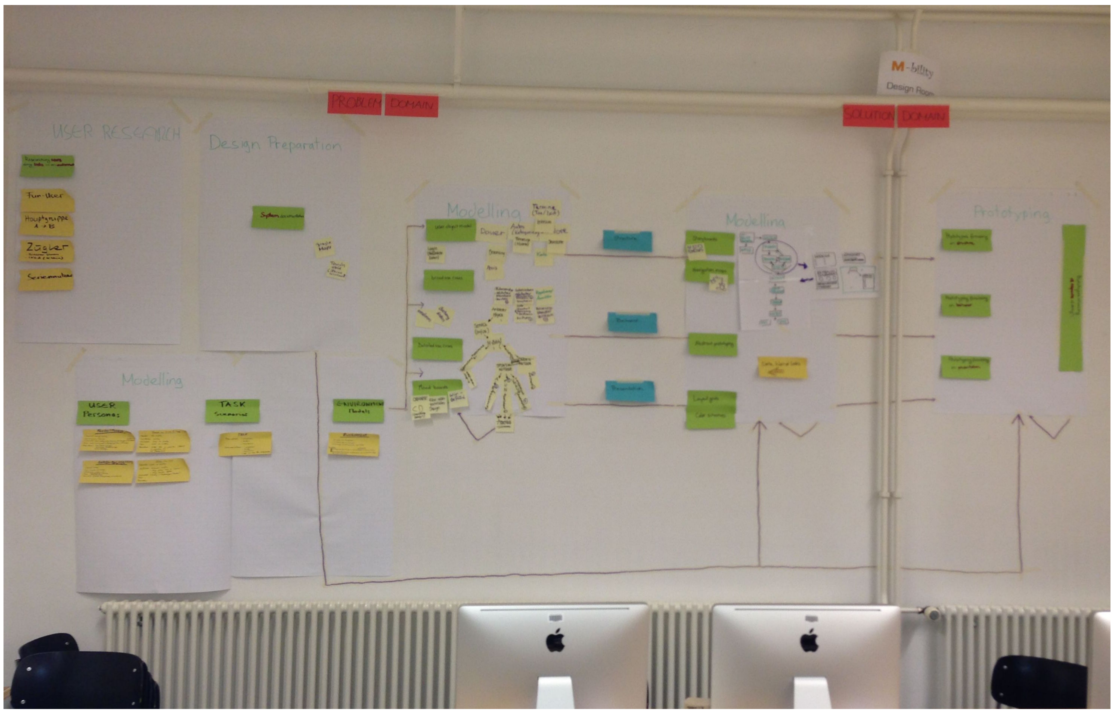


Figure 10. A wall in a design room of an educational project

5.3 Influence on the creation of interaction design pattern languages

For the construction of a formal interaction design pattern language, this work could mean the following: First, sufficient patterns are needed to cover the whole solution domain and level to which they belong must be clear (see Hübscher et al., 2011 for more on this topic). The second point is that the patterns have to "refer back" to all the different related aspects of the problem domain and to the levels of the solution domain that are prior to the one a pattern is on. All these relations have to be considered in the description of a pattern. If the patterns do not embody all these influences, the language might not be sufficiently differentiated.

6 Discussion

Here, some reflections will be made on the significance of this work together with a possible outlook.

6.1 The significance of this work

The mapping presented in section 4 shows that almost every aspect of the problem domain has an impact on various levels of the solution domain. For the task of designing a user interface, this means that on every level the designer needs to be informed about all relevant aspects of the problem domain. The results of this work can be seen as *cognitive artifact*s (Norman, 1991) supporting problem solving and/or learning of HCI practitioners and students. It can also support *reflection-in-action* (Schön, 1983; Löwgren and Stolterman, 2004) in order to improve one's interaction design skills. Furthermore, it can help to improve other UCD methods such as those for foundational user research, the setup of design rooms, and the development of interaction design pattern languages.

This work was initially developed to support the education of HCI students. The first author has used earlier versions of these ideas over several years in HCI education and it has been possible to evolve the ideas over time. This publication might pave the way to use the work even more in teaching and also in actual interaction design projects.

6.2 Outlook

Because this gap between user requirements and design of a user interface is very complex, it is easy to find several aspects to improve this work and therefore close the gap even further.

As discussed in Hübscher et al., (2011) the model proposed by Baxley (2002) could be made more complete and robust in various ways. It might also be worth trying to particularize Shackel's (2009) model even more. To have models of these two domains, which are very elaborate, would be of great benefit in other areas of HCI as well. The sub-categories of the four main aspects of Shackel's (2009) model introduced in section 2.1 have only been defined for the use in this work and not with a more general purpose in mind.

The examples of "patterns" mentioned in section 4 have been chosen purely to illustrate the various mappings. It would be interesting to have a complete pattern language to do the mapping with. The mapping could also be performed on the more detailed description of the solution domain with all the sub-layers mentioned by Baxley (2002).

The aspects that have been considered in Figure 9 are mainly *different phases* and *intermediate representations* in combination with the *mapping*. The aspects of *iteration* and *evaluation* have only considered marginally here. But because iteration and evaluation are very important topics in UCD, it would be very fruitful to include them in a more fundamental way. The mapping has only been made between the *models as a basis for the user interface*, the *user interface modeling*, and the *prototyping*. The gap between the *user and context models* and the *models as a basis for the user interface* is still unresolved thus a similar mapping could be done between these.

Acknowledgements

The authors would like to thank Zürcher Kantonalbank in Zurich, Switzerland for the support and funding of this research.

References

Apple (2016a). *Mac technology overview*. Retrieved May 14, 2016, from https://developer.apple.com/library/mac/documentation/MacOSX/Conceptual/OSX_Technology_Overview/

Apple (2016b). *OS X human interface guidelines*. Retrieved June 1, 2016, from https://developer.apple.com/library/mac/documentation/UserExperience/Conceptual/OSXHIGuidelines/

Arnowitz, J., Arent, M., & Berger, N. (2007). *Effective prototyping for software makers*. Elsevier, Amsterdam.

Bailey, B. P., Adamczyk, P. D., Chang, T. Y., & Chilson, N. A. (2006). A framework for specifying and monitoring user tasks. *Computers in Human Behavior, 22*(4): 709-732.

Baxley, B. (2002). *Making the web work: Defining effective web applications*. New Riders.

Baxley, B. (2003). Universal model of a user interface. In *Proceedings of the 2003 Conference on Designing for User Experiences* (pp. 1–14). ACM, New York.

Becker, S. A., & Mottay, F. E. (2001). A global perspective on web site usability. *Software, IEEE, 18*(1): 54-61.

Beyer, H. & Holtzblatt, K. (1998). *Contextual design: Defining customer-centered systems*. Morgan Kaufmann, San Francisco.

Brown, D. (2007). *Communicating design*. Peachpit Press, Berkeley.

Buxton, B. (2007). *Sketching user experiences*. Morgan Kaufmann, Amsterdam.

Carroll, J. M. (Ed.). (2000). *Making use: Scenario-based design of human-computer interactions*. The MIT Press, Cambridge, MA.

Chang, D., & Nesbitt, K. V. (2006). Developing gestalt-based design guidelines for multi-sensory displays. In *Proceedings of the 2005 NICTA-HCSNet Multimodal User Interaction Workshop-Volume 57* (pp. 9-16). Australian Computer Society, Inc.

Constantine, L. L. & Lockwood, L. A. D. (1999). *Software for use: A practical guide to the models and methods of usage-centered design*. Addison Wesley, Reading.

Cooper, A. (1999). *The inmates are running the asylum*. SAMS, Macmillan Computer Publishing, Indianapolis.

Cooper, A., Reimann, R., & Cronin, D. (2007). *About face 3: The essentials of interaction design*. Wiley Pub., Indianapolis.

Courage, C. & Baxter, K. (2004). *Understanding your users: A practical guide to user requirements*. Morgan Kaufmann, San Francisco.

Cutrell, E., & Guan, Z. (2007). What are you looking for? An eye-tracking study of information usage in web search. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 407-416). ACM.

Day, D., Lindgaard, G. & Noyes, J. (2009). Editorial: In Memoriam Brian Shackel 1927-2007. *Interacting with Computers, 21*(5-6): 324.

Dearden, A. & Finlay, J., 2006. Pattern languages in HCI: A critical review. *Human-Computer Interaction, 21*(1): 49-102.

Dey, A. K. (2001). Understanding and using context. *Personal and Ubiquitous Computing, 5*(1): 4-7.

Dirbach, J., Flückiger, M., & Lentz, S. (2011). *Software entwickeln mit Verstand*. dpunkt, Heidelberg.

Dow, S. P., Heddleston, K. & Klemmer, S.R. (2009). The efficacy of prototyping under time constraints. In *Proceedings of the Seventh ACM Conference on Creativity and Cognition* (C&C'09), (pp. 165–174). ACM, New York.

Eisenmann, T. R., Pao, M., & Barley, L. (2012). Dropbox: 'It Just Works'. *Harvard Business School Entrepreneurial Management Case,* (811-065).

Eisenstein, J., Vanderdonckt, J., & Puerta, A. (2000). Adapting to mobile contexts with user-interface modeling. In *Mobile Computing Systems and Applications, 2000 third IEEE workshop* (pp. 83-92). IEEE.

Endrissat, N. Islam, G. & Noppeney, C. (2016). Visual organizing: Balancing coordination and creative freedom via mood boards. *Journal of Business Research, 69*: 2353-2362.

Findlater, L., & McGrenere, J. (2004). A comparison of static, adaptive, and adaptable menus. In *Proceedings of the SIGCHI Conference on Human factors in Computing Systems* (pp. 89-96). ACM.

Fink, J., Kobsa, A., & Nill, A. (1997). Adaptable and adaptive information access for all users, including the disabled and the elderly. In *International Conference UM97*. (pp. 171-173). Springer, Wien.

Fouse, A., Weibel, N., Hutchins, E., & Hollan, J. D. (2011). ChronoViz: A system for supporting navigation of time-coded data. In *Proceedings of the 2011 Annual Conference Extended Abstracts on Human Factors in Computing Systems* (pp. 299-304). ACM.

Garrett, J. J. (2002). *The elements of user experience: User-centered design for the web*. New Riders, Indianapolis.

George, R. P., Anwar, R., & Jeyasekhar, S. (2011). Visual reading patterns on Arabic interfaces: Insights from eye tracking. *Journal of Computing, 3*(11).

Gerfelder, N., Spierling, U., & Müller, W. (2000). Novel user interface technologies and conversational user interfaces for information appliances. In *CHI'00 Extended Abstracts on Human Factors in Computing Systems* (pp. 41-42). ACM.

Goodwin, K. (2009). *Designing for the digital age*. Wiley, New York.

Google (2016). *Material design*. Retrieved June 1, 2016, from https://www.google.com/design/spec/material-design

Graefe, T. M. (1997). Transforming representations in user-centered design. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 57-79). CRC Press, Boca Raton.

Greenberg, S., Carpendale, S., Marquardt, N. & Buxton, B. (2012). *Sketching user experiences: The workbook*. Elsevier, Amsterdam.

Gruber, T. R. (1995). Toward principles for the design of ontologies used for knowledge sharing. *International Journal of Human-Computer Studies, 43*(5): 907-928.

Hackos, J. T. & Redish J. C. (1998). *User and task analysis for interface design*. John Wiley & Sons, New York.

Hall, S., Cockerham, K., & Rhodes, D. (2002). What's your color? [human-machine interface design]. *Industry Applications Magazine, IEEE, 8*(2): 50-54.

Heller, H., & Rivers, D. (1996). So you wanna design for the web. *Interactions, 3*(2): 19-23.

Horn, M. S., Crouser, R. J., & Bers, M. U. (2012). Tangible interaction and learning: The case for a hybrid approach. *Personal and Ubiquitous Computing, 16*(4): 379-389.

Houde, S., & Hill, C. (1997). What do prototypes prototype? In M. Helander, T. Landauer, & P. Prabhu (Eds.), *Handbook of human-computer interaction* (pp. 367–381). Elsevier Science, Amsterdam, 2nd edition.

Hübscher, C., Pauwels, S., Roth, S., Bargas-Avila, J. A., & Opwis, K. (2011). The organization of interaction design pattern languages alongside the design process. *Interacting with Computers, 23*(3): 189-201.

International Organization for Standardization (ISO). (2010). *ISO 9241-210: Human-centered design for interactive systems*. International Organization for Standardization, Genève.

Johnson, J. (2007). *GUI bloopers 2.0: Common user interface design don'ts and dos*. Morgan Kaufmann, San Diego.

Johnson, P., Johnson, H., & Hamilton, F. (2000). Getting the knowledge into HCI: Theoretical and practical aspects of task knowledge structures. In J. M. Schraagen, S. F. Chipman, & V. L. Shalin (Eds.), *Cognitive task analysis* (pp. 201-214). Lawrence Erlbaum and Associates, Mahwah.

Karahasanović, A., Brandtzæg, P., Heim, J., Lüders, M., Vermeir, L., Pierson, J., Lievens, B., Vanattenhoven, J., & Jans, G. (2009). Co-creation and user-generated content – elderly people's user requirements. *Computers in Human Behavior, 25*(3): 655-678.

Karat, J. & Bennett, J. L. (1991). Using scenarios in design meetings – a case study example. In J. Karat (Ed.), *Taking software design seriously* (pp. 63-94). Academic Press Professional, Inc., San Diego.

Kascak, L., Rébola, C. B., Braunstein, R., & Sanford, J. A. (2013). Icon design for user interface of remote patient monitoring mobile devices. In *Proceedings of the 31st ACM International Conference on Design of Communication* (pp. 77-84). ACM.

Kosters, G., Six, H. W., & Voss, J. (1996). Combined analysis of user interface and domain requirements. In *Requirements Engineering, 1996, Proceedings of the Second International Conference on* (pp. 199-207). IEEE.

Kunc, L., Macek, T., Labský, M., & Kleindienst, J. (2013). Speech-based text correction patterns in noisy environment. In *Human-Computer Interaction. Interaction Modalities and Techniques* (pp. 59-66). Springer, Berlin.

Larman, C., & Basili, V. R. (2003). Iterative and incremental developments. A brief history. *Computer, 36*(6): 47-56.

Lavie, T., & Meyer, J. (2010). Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies, 68*(8): 508-524.

Lim, Y. K., Stolterman, E. & Tenenberg, J. (2008). The anatomy of prototypes: Prototypes as filters, prototypes as manifestations of design ideas. *ACM Transactions on Computer-Human Interaction (TOCHI), 15*(2): 7.

Linderman, M., & Fried, J. (2004). *Defensive design for the web: How to improve error messages, help, forms, and other crisis points*. New Riders Publishing.

Löwgren, J. & Stolterman, E. (2004). *Thoughtful interaction design: A design perspective on information technology*. The MIT Press, Cambridge.

Ludolph, F. (1997). Model-based user interface design: Successive transformations of a task/object model. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 81-107). CRC Press, Boca Raton.

Lunn, D., & Harper, S. (2011). Providing assistance to older users of dynamic web content. *Computers in Human Behavior, 27*(6): 2098-2107.

Maguire, M., & Bevan, N. (2002). User requirements analysis. In J. Hammond, T. Gross & J. Wesson (Eds.), *Usability: Gaining a competitive edge* (pp. 133-148). Springer US.

Mankoff, J., Hudson, S. E., & Abowd, G. D. (2000). Interaction techniques for ambiguity resolution in recognition-based interfaces. In *Proceedings of the 13th Annual ACM Symposium on User Interface Software and Technology* (pp. 11-20). ACM.

Matrai, R. (Ed.) (2010). *User interfaces*. InTech, Rijeka, Croatia.

Mayhew, D. J. (1992). *Principles and guidelines in software user interface design*. Prentice Hall, Englewood Cliffs.

Mayhew, D. J. (1999). *The usability engineering lifecycle: A practitioner's handbook for user interface design*. Morgan Kaufmann Publishers, San Francisco.

McCurdy, M., Connors, C., Pyrzak, G., Kanefsky, B., & Vera, A. (2006). Breaking the fidelity barrier: An examination of our current characterization of prototypes and an example of a mixed-fidelity success. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (pp. 1233–1242). ACM.

Microsoft (2016). *Design applications for the Windows desktop*. Retrieved June 1, 2016, from https://developer.microsoft.com/en-us/windows/desktop/design

Nielsen, J. (1993). *Usability engineering*. Academic Press, Boston.

Norman, D. A. (1991). Cognitive artifacts. In J. M. Carroll (Ed.), *Designing interaction: Psychology at the human-computer interface* (pp. 17-38). Cambridge University Press.

Peissner, M., Schuller, A., & Spath, D. (2011). A design patterns approach to adaptive user interfaces for users with special needs. In *Human-Computer Interaction. Design and Development Approaches* (pp. 268-277). Springer, Berlin.

Perkins, R., Keller, D. S., & Ludolph, F. (1997). Inventing the Lisa user interface. *Interactions, 4*(1), 40-53.

Pfeil, U., Arjan, R., & Zaphiris, P. (2009). Age differences in online social networking – a study of user profiles and the social capital divide among teenagers and older users in MySpace. *Computers in Human Behavior, 25*(3): 643-654.

Rantzer, M. (1996). The delta method – a way to introduce usability. In D. Wixon & J. Ramey (Eds.), *Field methods casebook for software design* (pp. 91–112). Wiley Computer Pub., New York.

Rantzer, M. (1997). Mind the gap: Surviving the dangers of user interface design. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 153-184). CRC Press, Boca Raton.

Roberts, D., Berry, D., Isensee, S., & Mullaly, J. (1998). *Designing for the user with OVID: Bridging user interface design and software engineering*. Software engineering series. Macmillan Technical Pub., Indianapolis.

Rohlfs, S. (1997). Transforming user-centered analysis into user interface: The redesign of complex legacy systems. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 185-214). CRC Press, Boca Raton.

Roth, S. P., Schmutz, P., Pauwels, S. L., Bargas-Avila, J. A., & Opwis, K. (2010). Mental models for web objects: Where do users expect to find the most frequent objects in online shops, news portals, and company web pages? *Interacting with Computers, 22*(2): 140-152.

SAP (2016). *SAP design guidelines and resources*. Retrieved June 1, 2016, from https://experience.sap.com/guidelines/

Schleicher, R., Shirazi, A. S., Rohs, M., Kratz, S., & Schmidt, A. (2011). WorldCupinion experiences with an android app for real-time opinion sharing during soccer world cup games. *International Journal of Mobile Human Computer Interaction (IJMHCI), 3*(4): 18-35.

Schmidt, A., Beigl, M., & Gellersen, H.-W. (1999). There is more to context than location. *Computers & Graphics, 23*(6): 893-901.

Scholtz, J., & Salvador, T. (1997). Systematic creativity: A bridge for the gaps in the software development process. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 215-244). CRC Press, Boca Raton.

Schön, D. A. (1983). *The reflective practitioner: How professionals think in action*. Basic Books, New York.

Shackel, B. (2009). Usability – context, framework, definition, design and evaluation. *Interacting with Computers, 21*(5-6): 339-346.

Shneiderman, B. & Plaisant, C. (2004). *Designing the user interface: Strategies for effective human-computer interaction*. Addison Wesley, Boston, 4th edition.

Simpson, K. T. (1997). The UI war room and design prism: A user interface design approach from multiple perspectives. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 245-274). CRC Press, Boca Raton.

Snyder, C. (2003). *Paper Prototyping the fast and easy way to design and refine user interfaces*. Morgan Kaufmann, Elsevier Science, San Francisco.

Taslim, J., Adnan, W. A. W., & Bakar, N. A. A. (2009). Investigating children preferences of a user interface design. In *Human-Computer Interaction. New Trends* (pp. 510-513). Springer, Berlin.

Thalheim, B., & Düsterhöft, A. (2000). The use of metaphorical structures for internet sites. *Data & Knowledge Engineering, 35*(2): 161-180.

Tidwell, J. (2006). *Designing interfaces*. O'Reilly, Beijing.

U. S. Dept. of Health and Human Services. (2006). *The research-based web design & usability guidelines*. U.S. Government Printing Office, Washington, enlarged/expanded edition.

Van Duyne, D. K., Landay, J. A., & Hong, J. I. (2007). *The design of sites: Patterns for creating winning web sites*. Prentice Hall, Upper Saddle River, 2nd edition.

Van Harmelen, M. (Ed.). (2001). *Object modeling and user interface design: Designing interactive systems*. Addison-Wesley Longman, Inc., Reading.

Van Welie, M. (2009). *Patterns in interaction design*. Retrieved May 7, 2009, from http://www.welie.com/patterns/

Van Welie, M. & van der Veer, G. C. (2003). Pattern languages in interaction design: Structure and organization. In *Proceedings of Interact*, *vol 3*: 1-5.

Vu, K. P. L., & Proctor, R. W. (Eds.). (2011). *Handbook of human factors in web design*. CRC Press, Boca Raton.

Warfel, T. (2009). *Prototyping: A practitioner's guide*. Rosenfeld Media, Brooklyn.

Wiegers, K. & Beatty, J. (2013). *Software requirements*. Microsoft, Redmond.

Wood, L. E. (1997a). *User interface design: Bridging the gap from user requirements to design*. CRC Press, Boca Raton.

Wood, L. E. (1997b). Introduction: Bridging the design gap. In L. E. Wood (Ed.), *User interface design: Bridging the gap from user requirements to design* (pp. 1-14). CRC Press, Boca Raton.

Wu, J.-H., & Yuan, Y. (2003). Improving searching and reading performance: The effect of highlighting and text color coding. *Information & Management, 40*(7): 617-637.

Yadav, S. B. (2010). A conceptual model for user-centered quality information retrieval on the world wide web. *Journal of Intelligent Information Systems, 35*(1): 91-121.

Yahoo! Inc. (2009). *Design pattern library*. Retrieved May 29, 2009, from http://developer.yahoo.com/ypatterns/

Young, I. (2008). *Mental models: Aligning design strategy with human behavior*. Rosenfeld Media, New York.

Zhang, Y. (2008). The influence of mental models on undergraduate students' searching behavior on the web. *Information Processing & Management, 44*(3): 1330-1345.