

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).

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 changing 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 main 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 an ordered list and frequency numbers but that does not facilitate easy comparisons 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 apparent 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 simplistic, 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.

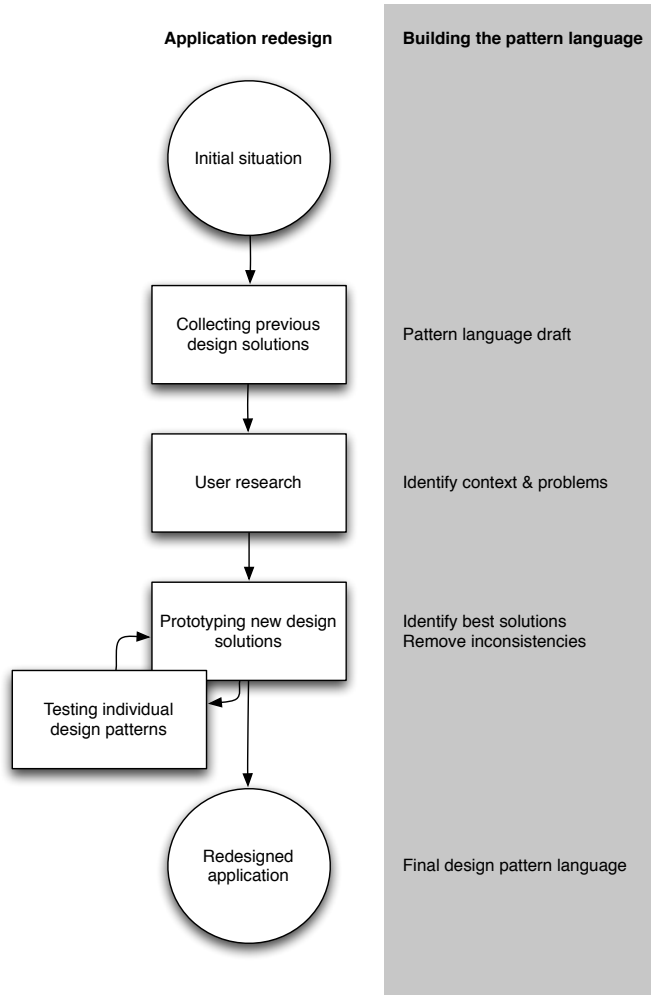
Our hypothesis is that building a complete, validated pattern language for a specific domain takes four basic steps (see figure 3) that addresses all the important pattern language issues:

- (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.



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 han-

dling 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

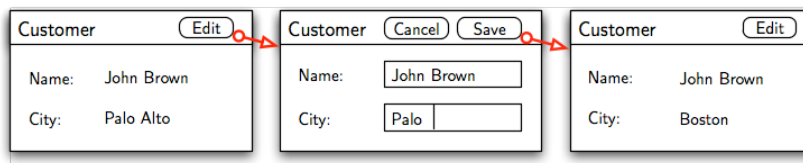
- usability problems was redesigned.
- (2) Creating new interaction design patterns: Discovered problems that were not addressed in the application’s previous version were solved and added to the design pattern language.
 - (3) Removing interaction design patterns: Design solutions that were obsolete 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.

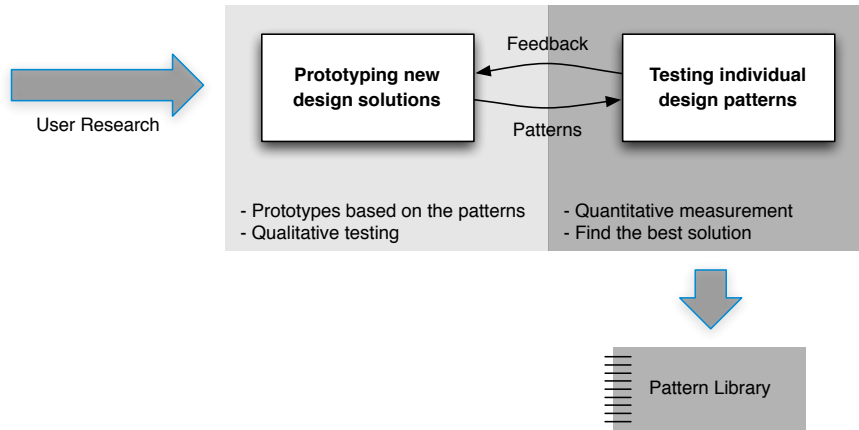
A pattern was included in testing if one of the following cases occurred:

- (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.



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

Tested pattern	Number of participants					Total
	Phase 1	Phase 2	Phase 3	Phase 4	Phase 5	
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.

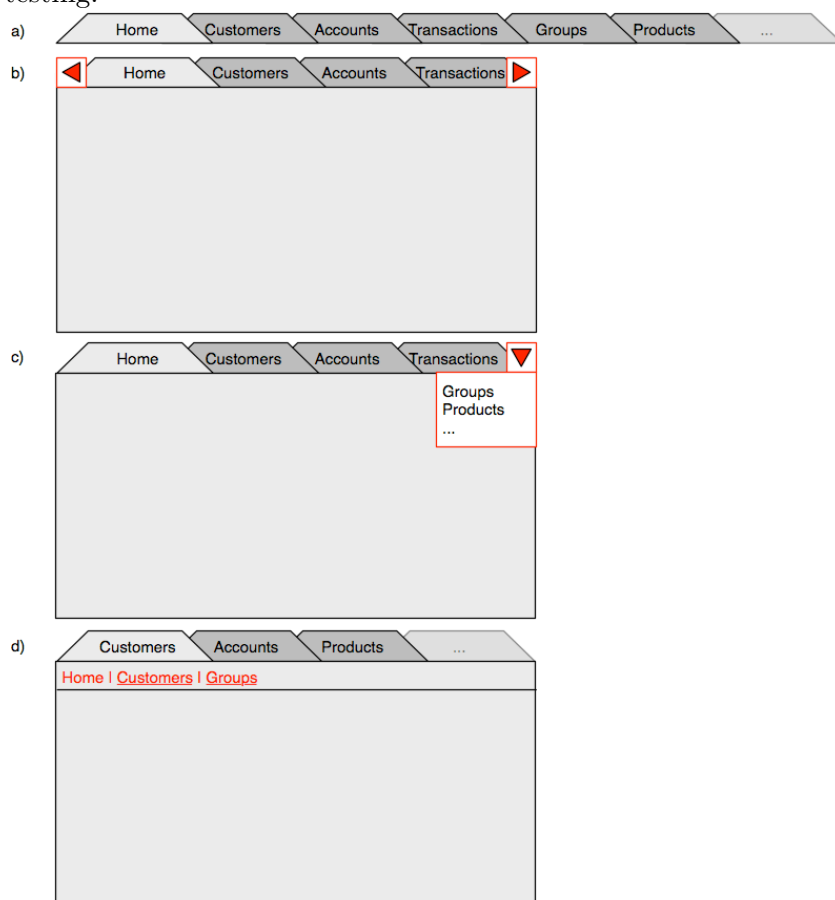
Let us consider the main navigation (figure 6) as an example: It was identified as a problem in user research (see section 3.2). The redesigned application still has a considerable number of sections (figure 6.a). To prevent people from getting lost scrolling through the tabs as in the previous design (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.

Measures	n	Main navigation design	
		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, Hyper-link
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.

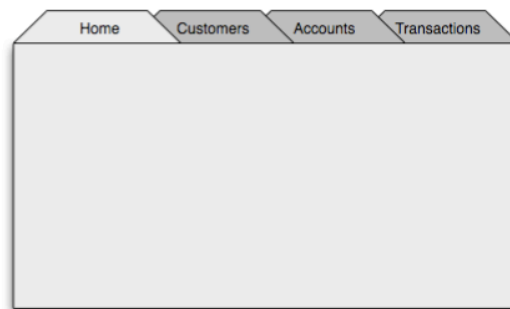
We show that completeness of a domain-specific pattern language can be achieved during a user-centered redesign project. Mining an application’s previous version systematically for all solutions that it contains, as well as documenting all changes on pattern level guarantees morphological completeness, 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-specificity, 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.
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

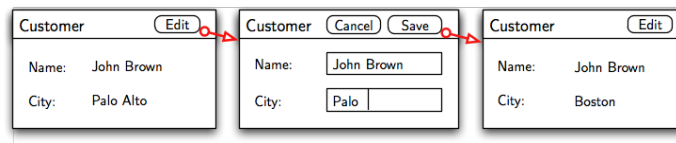
- Tabs

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?

Whenever 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., 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 Interaction Conference. 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>