

Chapter 2

From HCI Patterns Languages to Pattern-Oriented Design

Abstract During last decade, several human–computer interaction (HCI) researchers and practitioners introduced their own pattern languages with specific terminology and classification. Pattern languages have been considered as a lingua franca for crossing cultural and professional barriers between different stakeholders. Pattern languages have also been presented as building blocks at different levels of granularity, which can be combined to compose new interactive systems. Despite the obvious and acclaimed potential for supporting design, patterns languages has not achieved the acceptance and widespread applicability envisaged by their authors. This chapter provides an analysis of the facts about pattern languages and pattern-based design approaches. Some shortcomings in the presentation and application of HCI patterns languages are identified and discussed under the prevailing fallacies. Based on the analysis of how pattern languages have been used so far, we draw some recommendations and future perspectives on what can be done to address the existing shortcomings. Making pattern languages more accessible, easily understandable, comparable and integratable in software, and usability in engineering tools can promote HCI patterns to claim the usability, usefulness, and importance originally envisaged for the pattern-oriented design approach.

2.1 Patterns as Tool to Capture Design Knowledge and Best Practices

Historically, best practices reusability in human–computer interaction (HCI) has attracted far less attention in comparison with other disciplines like software engineering, but this trend has been changing. The user interface (UI) occupies a large share of the total size of a typical system (Myers et al. 1993), and the design of interactive systems can be facilitated by applying best design practices. In current practice, tools for capturing and disseminating design knowledge include guidelines, claims and patterns (Macintosh 1992; Microsoft 1995), and within organizations (Billingsley 1995; Rosenzweig 1996; Weinschenk and Yeo 1995). Guidelines concentrate most often on the physical design attributes of the user interface, and examples are the *Macintosh Human Interface Guidelines* (Macintosh 1992) and the *Java Look and Feel Design Guidelines* (Sun Microsystems 2001). Claims (Sutcliffe 2000) are

Table 2.1 Reused claim for a safety-critical application

Reused claim: safety-critical application
<i>Claim ID:</i> rare event monitor
<i>Target artifact:</i> user interface for a chemical analysis instrument control system
<i>Description:</i> infrequent, dangerous events are detected by the system and a warning is issued to the user; in this case operational failures in a laser gas chromatograph control system
<i>Upside:</i> automatic detection of dangerous events relieves the user of constant monitoring; automatic detection and warning gives the user time to analyze the problem
<i>Downside:</i> issuing too many warnings may lead the user to ignore critical events; automated monitoring may lead to user’s overconfidence in the automated system and decrease their situation awareness
<i>Scenario:</i> no events are detected in the laser emission controller or power supply, so the system gives an audio warning to the user and visually signals the location of the problem on a diagram of the instrument

a means to capture HCI knowledge in association with a specific artefact and usage context. They provide design advice based on theoretical foundations, cognitive design rationale, and possible trade-offs.

In the 1990s, design guidelines became an increasingly popular way to disseminate usability knowledge and ensure a degree of consistency across applications (Macintosh 1992; Microsoft 1995) and within organizations (Billingsley 1995; Rosenzweig 1996; Weinschenk and Yeo 1995). These guidelines often took the form of style guides and were usually platform-specific, prescribing how different kinds of windows should look and interact with the user for tasks such as choosing from lists or menu controls. An example of a *Java Look and Feel* guideline for a toolbar is described in Table 2.1. To date, guidelines have yet to realize their full potential and have had little impact on the design of user interface software (Gould et al. 1991; De Souza and Bevan 1990). Apart from not adequately addressing concerns facing designers such as which guidelines should be used under what circumstances (Henninger et al. 1995), studies have shown that interface guidelines suffer from being too abstract to be applied directly (Tetzlaff and Schwartz 1991; Thovtrup and Nielsen 1991). Most guidelines fall short of the goal of putting the accumulated knowledge of user-centered design at the fingertips of everyday designers, often becoming a static document read only by human factors specialists.

Introduced in the last decade, Claims (Sutcliffe 2000) are another means to capture and disseminate HCI design knowledge. They are associated with a specific artefact and usage context, providing design advice and possible trade-offs. Claims are powerful tools because, in addition to providing negative and positive design implications, they contain both theoretical and cognitive rationales. They also contain associated scenarios that provide designers with a concrete idea of the context of use. When first introduced, claims were limited in their generality as they were too narrowly defined with specific scenarios and examples. Subsequently, the paradigm of reuse was applied to claims in order to make them more generic and

applicable to a wider range of application contexts. An example of such a *reused claim* for a safety-critical application is given in Table 2.1.



Although both guidelines and claims promote reuse, they have yet to be adopted by the mainstream designer. Studies have shown that interface guidelines suffer from being too abstract to directly apply (Tetzlaff and Schwartz 1991; Thovtrup and Nielsen 1991), while claims are too grounded in specific scenarios and examples, limiting their generality (Sutcliffe 2000).

HCI design patterns only capture essential details of design knowledge, and abstract away from superfluous, toolkit-dependent, and platform-specific design information. In addition, the presented information is organized intuitively within a set of predefined attributes, allowing designers, for example, to search rapidly and effectively through different design solutions while assessing the relevance of each pattern to their design. Every pattern has three necessary elements, usually presented as separate attributes, which are: a context, a problem, and a solution. The context describes a recurring set of situations in which the pattern can be applied. The problem refers to a set of forces, i.e., goals and constraints, which occur in the context. The solution refers to a design form or a design rule that can be applied to resolve the problem. The solution describes the elements that constitute a pattern, the relationships between these elements, as well as their responsibilities and collaboration. Other attributes that may be included are additional design rationale, specific examples, and related patterns.

Patterns alleviate many of the shortcomings associated with guidelines. Above all, they are a good alternative to guidelines as they are problem-oriented, but not platform-specific. Their descriptive format, with the use of defined attributes, is more concrete and easier to apply for novice designers. Guidelines can be quite abstract and intangible when it comes to practical application, whereas patterns are more structured and the knowledge is placed in a context. The designer is told when, how, and why the solution can be applied. Since patterns are context-oriented, the solution is related to a specific user activity. Table 2.2 compares a guideline and a pattern that addresses the same problem: helping the user to find frequently used commands or pages. The pattern version of the description gives detailed information about the context in which the solution can be applied.

Patterns have a more complementary association with claims; this in contrast to their somewhat antagonistic relationship with guidelines. Claims are tightly bound to specific domains of use, but contain valuable information including design trade-offs, and a possibility is to use them to complement patterns creating a “package of reusable knowledge” (Sutcliffe 2000). Such detailed information can be incorporated when the pattern is instantiated to a specific context of use. Furthermore, details from claims about design and cognitive rationale, including scenario descriptions, can provide additional information to designers when combining patterns to create comprehensive designs.

Table 2.2 Guideline versus pattern for the toolbar

Guideline	Pattern
<p>A toolbar is a collection of frequently used commands or options that appear as a row of toolbar buttons.</p> <p>Toolbars normally appear horizontally beneath a primary window's menu bar, but they can be dragged anywhere in the window or into their own window.</p> <p>Toolbars typically contain buttons, but you can provide other components (such as text fields and combo boxes) as well.</p> <p>Toolbar buttons can contain menu indicators, which denote the presence of a menu. Toolbars are provided as shortcuts to features available elsewhere in the application, often in the menus.</p>  <p>Source: Java Look and Feel Design Guidelines (Sun Microsystems, 2001).</p>	<p>Pattern Name: Convenient Toolbar</p> <p>Type: Navigation Support</p> <p>Context</p> <ul style="list-style-type: none">– Task: Assist the user to reach convenient and key web pages at any time– User: Novice or expert– Environment: Website <p>Problem</p> <ul style="list-style-type: none">– Help the user find useful and “safe” pages that need to be accessed from any location on the site, regardless of the current state of the artefact.– The user should reach these pages promptly. <p>Solution</p> <ul style="list-style-type: none">– Group the most convenient action links such as home, site map, and help– Use meaningful metaphors and accurate phrases as labels.– Place them consistently throughout the website <p>Other attributes</p> <ul style="list-style-type: none">– Specific Forces, Related Patterns, Design Rationale <p>Specific example</p> 

2.2 HCI Design Pattern Languages

Patterns have been organized into pattern languages. Just as words must have grammatical and semantic relationships with each other in order to create sentences with meaning, design patterns must be related to each other in order to form meaningful design constructs. Pattern languages are a structured method of describing good design practices, containing a collection of interrelated patterns that aim to disseminate the body of contained knowledge. For designers, pattern languages are a means to traverse common HCI problems in a logical way, describing the key characteristics of effective solutions for meeting various design goals. Furthermore, they act as a communicative design tool and give rise to many different paths through the design activity.

A number of pattern languages have been suggested in HCI. For example, (Duyne 2003) “The Design of Sites,” (Welie 1999) Interaction Design Patterns, and (Tidwell 1997) UI Patterns and Techniques play an important role. In addition, specific languages such as (Laakso 2003) User Interface Design Patterns and the UPADE Language (Engelberg and Seffah 2002) have been proposed as well. Different pattern collections have been published including patterns for web page layout design (Tidwell 1997) and (Coram and Lee 1998) for navigation in large information architectures, as well as for visualizing and presenting information.

Pattern languages have three essential elements. First, the language has to contain a standard pattern definition. One format for defining patterns was presented in the previous section (Table 2.1 and 2.2)—with the common attributes context, problem, solution, forces, related patterns, and examples. Second, the language must logically group patterns. Tidwell (1997) organizes her patterns according to different facets of UI design; categories include content organization, navigation, page layout, and actions/commands. Another example is the experiences pattern language (Fig. 2.1) developed by (Coram and Lee 1998), which concentrates on the user’s experience within software systems. The main focus is on the interactions between the user and the interfaces of software applications. Patterns are grouped according to different focus areas and user interface paths such as interaction style, explorable interface, and symbols. Third, pattern interrelationships should be described. In experiences,

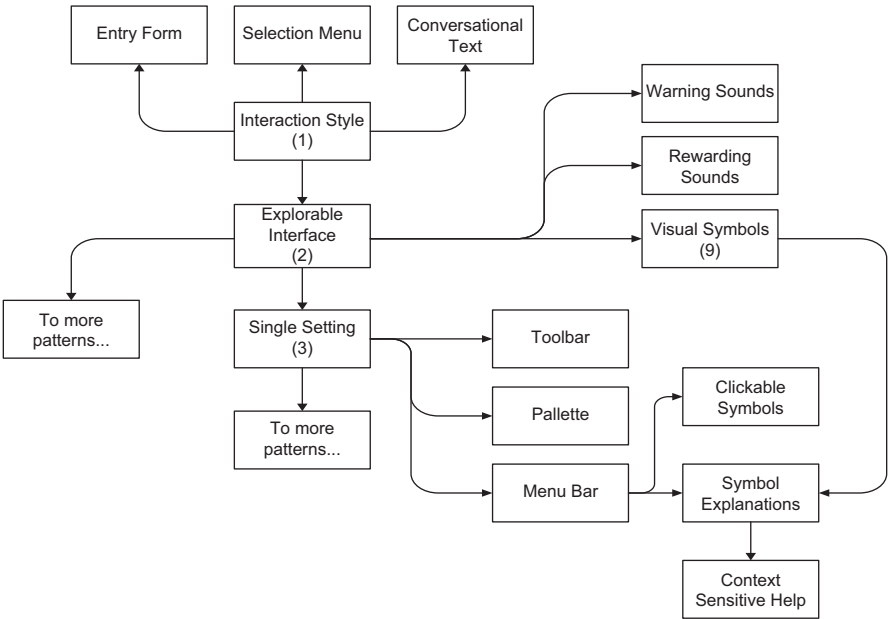


Fig. 2.1 The experiences pattern language

the relationships between the patterns are mapped and indicated by arrows, creating a sort of “flow” within the language. This is illustrated in Fig. 2.1.

Distinguishing between different types of relationships reinforces the generative nature of pattern languages, and supports the idea of using patterns to develop complete designs. However, for designers to be able to use patterns effectively and with efficacy to solve problems in HCI and interactive system design, patterns need to be intimately related to a design process. Based on the design problem, pattern languages should provide starting points for the designer, and a means to systematically walk the designer from pattern to pattern.

For example, in experiences, the metapattern *Interaction style* (denoted with “(1)” in Fig. 2.1) is the first pattern that leads the designer along the major paths through the language. The design advice (Coram and Lee 1998) for this pattern includes studying the user and environment, working with the user to determine what interaction style is best, and keeping the interface simple and consistent. This pattern is connected to four other patterns as indicated by arrows in Fig. 2.1 (entry form, selection menu, conversational text, and *explorable interface*). Based on the context of use, the designer is free to choose any of these patterns to incorporate into the design. This is a repetitive process as some patterns, such as *Explorable Interface*, are subsequently connected to even more suggested patterns.

Although the experiences language showed the beginnings of associating its patterns to a design process, it was regrettably not developed in its entirety. In the next section, we will present some attempts at further linking pattern languages to the UI design process.

Having studied linguistics and psycholinguistics extensively, we have some difficulty with the fact that some pattern authors call their works languages. In most cases they are merely collections or taxonomies. Even Alexander’s own original work is disappointing by the standard of what we know about languages. At best these collections offer some primitive aspects of the combinatorial generativity that we see in natural language, but they are severely lacking in the syntactical and grammatical properties that are necessary for a language. The fact that we can combine two patterns together is not sufficient to make the pattern collection into a language. This weakness in the current pattern “languages” points to an enormous uncharted area of R&D for turning these collections into true languages. This language aspect of patterns is precisely what is needed for making model-driven approaches useful in practice.

2.3 HCI Pattern Languages and the User-Centered Design Process

The interface design of an interactive system can be a challenging task—and especially so when a project involves different design participants and stakeholders. Successful designs require individuals to communicate their concepts and ideas, building a common forum for the discussion of already available design practices.

As in any culture or society, the HCI community needs a common ground for such communication and dissemination of knowledge. Designers focus on the creation of an artifact that integrates various behavioral theories and technologies. This is done without regard to the evaluation of individual variables that may affect the design (Zimmerman et al. 2004). Usability experts take a more scientific approach, looking at specific behavioral and design elements that best satisfy the requirements. Software developers are interested in finding an applicable design and implementing it correctly in the most efficient manner, and are often not familiar with usability engineering techniques and human interaction theories (Myers and Rosson 1992).

This is a proving ground for patterns as they provide a mechanism to successfully integrate and satisfy the different goals of all individuals involved in the design process, crossing cultural and professional barriers, and overcoming limitations in communication. Patterns are presented consistently, are easy to read, and provide background reasoning. They act as a lingua franca (Erickson 2000) for design, which can be read and understood by all. (Erickson 2000) discusses the potential of this as a way of making communication in design a more “egalitarian process,” with the focus relying less on technical design issues, and more upon broader design problems and solutions. A lingua franca facilitates discussion, presentation, and negotiation for the many different individuals who play a role in designing interactive systems.

Acting as a communicative vehicle, pattern languages are interesting tools that can guide software designers through the design process. However, there exists no commonly agreed upon UI design process that employs pattern languages as first class tools. Several people have tried to link patterns to a process or framework, bringing some order to pattern languages, and suggesting that potentially applicable patterns be identified early on based on user, task, and context requirements. A pattern-driven design process should lead designers to relevant patterns based on the problem at hand, demonstrate how they can be used, as well as illustrate combinations with related patterns. In the following section, we describe three design approaches driven by patterns.

In the pattern-supported approach (PSA) framework (Fig. 3.2), HCI patterns are used at various levels to solve problems related to business domains and processes, tasks, structure and navigation, and graphical user interface (GUI) design (Granlund and Lafrenière 1999). The main idea that can be drawn from PSA is that HCI patterns can be documented identified and instantiated according to different parts of the design process—giving us knowledge as early on as during system definition. For example, during system definition or task and user analysis, depending on the context of use, we can decide which HCI patterns are appropriate for the design phase. Although PSA shows the beginnings of associating patterns to the design process, pattern interrelationships and their possible impact on the final design are not tackled in detail.

Duyne et al. (2003) describe a second approach, where patterns are arranged into 12 groups that are available at different levels of web design. Their pattern language has 90 patterns that address various aspects of web design, ranging from creating a navigation structure to designing effective page layouts. The order of their pattern

Table 2.3 Pattern groups ordered according to a web design process

Step	Pattern groups	Description	Pattern examples
A	Site genres	Construct particular site type	Personal e-commerce Nonprofits as networks of help
B	Creating a navigation framework	Choose patterns to navigate, browse and search on the site	Multiple ways to navigate Task-based organization
C	Creating a powerful homepage	Design the homepage based on user needs	Homepage portal Up-front value proposition
D	Writing and managing content	Manage content and address user accessibility	Page templates Internationalized and local content
E	Building trust and credibility	Address issues dealing with trust and credibility	Site branding Fair information practices
F	Basic e-commerce	Create a good customer experience for e-commerce	Quick-flow checkout Clean product details
G	Advanced e-commerce	Incorporate advanced e-commerce features	Featured products Cross-selling and up-selling
H	Helping customers complete tasks	Structure your site to improve task completion	Process funnel Persistent customer sessions
I	Designing effective page layouts	Create clear, predictable and understandable layouts	Grid layout Expanding width screen size
J	Making site search fast and relevant	Design interaction so that user searches are effective	Search action module Straightforward search forms
K	Making navigation easy	Display helpful navigation elements	Unified browsing hierarchy Action buttons
L	Speeding up your site	Incorporate patterns to make your site look and feel fast	Low number of files Fast downloading images

groups generally indicates the order in which they should be used in the design process (Table 2.3). In addition, patterns chosen from the various groups have links to related patterns in the language. The highest level pattern group in their scheme is *Site Genres*, which provides a convenient starting point into the language, allowing the designer to choose the type of site to be created. Starting from a particular Site Genre pattern, various lower level patterns are subsequently referenced. In this way, the approach succeeds not only in providing a starting point into the language, but also demonstrates how patterns of different levels may interact with one another.

2.4 Pattern Supported Approach (PSA)

The “Pattern Supported Approach” (PSA) addresses patterns not only during the design phase, but also during the entire software development process. PSA (Granlund et al. 2001) aims to support early system definition and conceptual design through the use of HCI patterns. In particular, patterns have been used to describe business domains, processes, and tasks to aid early system definition and conceptual design.

The main idea of PSA is that HCI patterns can be documented according to the development lifecycle. In other words, during system definition and task analysis, depending on the context of use, it can be decided which HCI patterns are appropriate for the design phase. In contrast to POD, the concept of linking patterns together to result in a design is not tackled in this approach.

The PSA to the user interface design process suggests a wider scope for the use of patterns by looking at the overall design process. Based on the fact that the usability of a system emerges as the product of the user, the task and the context of use, PSA integrates this knowledge in most of its patterns, dividing the forces in the pattern description correspondingly (i.e., describing task, user, and context forces). PSA provides a double-linked chain of patterns (parts of an emerging pattern language) that *support* each step of the design process (Granlund et al. 2001).

Building on PSA, PSA-proposed approach highlights another important aspect of pattern-oriented design: *pattern combination*. By combining different patterns, developers can use pattern relationships and combine them in order to produce an effective design solution. As a result, patterns become a more effective vehicle that supports design reuse.

Up to this point, most of the work on patterns in HCI has focused on screen design issues. PSA addresses patterns not only at the design phase, but also *before* design (Fig. 2.2).

For example, *task* patterns point to *Structure and Navigation Patterns*, which in turn point to *GUI Design Patterns*, and vice-versa. These patterns offer a way to capture and communicate knowledge from previous designs (including the knowledge from system definition, task/user analysis and structure and navigation design). Given a mature language of patterns belonging to the described classes, the PSA approach provides an entry point to this pattern language, and suggests (without restricting the pattern usage) a chain of appropriate patterns at different levels of analysis and design (Granlund et al. 2001).

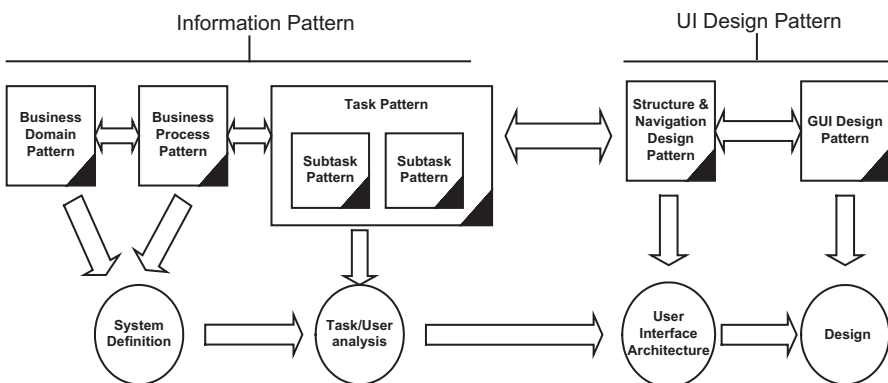


Fig. 2.2 The PSA framework with the relationships between PSA patterns

2.5 Pattern-Oriented Design

Javahery and Seffah (2002) proposed first the design approach called Pattern-Oriented Design (POD). The initial motivation for POD arose from interviews carried out with software developers using our patterns from the UPADE web language. These interviews revealed that in order for patterns to be useful, developers need to know how to combine them to create complete or partial designs. Providing a list of patterns and loosely defined relationships, as is the case for most HCI pattern languages, is insufficient to effectively drive design solutions. Understanding when a pattern is applicable during the design process, how it can be used, as well as how and why it can or cannot be combined with other related patterns, are key notions in the application of patterns.

First, POD provides a framework for guiding designers through stepwise design suggestions. At each predefined design step, designers are given a set of patterns that are applicable. This is in stark contrast to the current use of pattern languages, where there is no defined link to any sort of systematic process. Pattern relationships are explicitly described, allowing designers to compose patterns based on an understanding of these relationships.

As a practical illustration, we have applied POD within the context of the UPADE pattern language for web design. Each pattern in UPADE provides a proven solution for a common usability and HCI-related problem occurring in a specific context of use for web applications. Patterns are grouped into three categories, corresponding closely to the various steps and decisions during the process of web design: architectural, structural, and navigation support. Structural patterns are further subcategorized into page manager and information container patterns (Fig. 2.3 for pattern examples). During each design step, designers choose from a variety of applicable patterns: (1) architectural, relating to the architecture of the entire website; (2) page manager, establishing the physical and logical screen layout; (3) information container, providing ways to organize and structure information; and (4) navigation support, suggesting different models for navigating between information segments and pages.

Taleb et al. (2006) have described five types of relationships between categories patterns. This multicriterion classification is based on the original set of relationships (Zimmer 1994; Duyne et al. 2003; Yacoub and Ammar 2003) used to classify the patterns proposed in (Gamma et al. 1995). The relationships are used to compose a UI design, allowing designers to make suppositions such as: for some problem P, if we apply pattern A, then patterns B and C apply as subordinates, but pattern D cannot apply since it is a competitor. The relationships are explained below:

In POD, designers first should follow a POD model. We have published literature on a preliminary version of this model (Javahery and Seffah 2002). As part of this thesis, we refined the model and the corresponding pattern relationships. POD defines the overall design composition of a particular type of application, including a breakdown of this composition into different UI facets. The model acts as a guide for designers in making stepwise design decisions. To illustrate, for website design,

Patterns of HCI Design and HCI Design of Patterns
Bridging HCI Design and Model-Driven Software
Engineering

Seffah, A.

2015, XVII, 272 p. 161 illus., Hardcover

ISBN: 978-3-319-15686-6