

Tracing the evolution of HCI patterns as an interaction design tool

Ahmed Seffah · Mohamed Taleb

Received: 28 November 2008 / Accepted: 3 December 2011
© Springer-Verlag London Limited 2011

Abstract Design patterns have been introduced as a medium to capture and disseminate the best design knowledge and practices. In the field of human–computer interaction, practitioners and researchers have explored different avenues to use patterns and pattern languages as design tools. This paper surveys these avenues—from individual pattern use for solving a specific design problem, to pattern-oriented design, which guides designers in building a conceptual design by leveraging relationships between patterns. One of our underlying goals is to investigate how patterns can be used, not only to foster the reuse of proven and valid design solutions, but also as a central artefact in the process of deriving a design from user experiences and requirements. We will present our investigations on pattern-based design, and discuss how user experiences can be incorporated in the pattern selection process through the use of user variables, pattern attributes and associated relationships.

Keywords Engineering · Evolution · Visualization · Pattern · Usability · User experience

1 Introduction

In the field of interactive systems design defined in Sect. 2.2, reinventing the “wheel” when designing is time-consuming and ineffective. The capture and use of previous design

knowledge and best practices is becoming a fundamental mission of human–computer interaction (HCI) [1]. For example, designers need to be aware of the many different factors and context that influence UI design defined in Sect. 3, such as user requirements, environmental constraints, platform and device characteristics, and task analysis in the intended place of use. Furthermore, since the UI design process is context sensitive, UI designs based on the intuition of the designer are no longer adequate. In order to cope with this multi-dimensional complexity, we require design tools that are systematic, traceable and practical for designers, but which also leave room for design creativity when appropriate.

Although interactive systems logic engineering has become increasingly driven by rigorous techniques and principles, the design and engineering of the UI has remained “ad hoc” and thus has resulted—from a software engineering point of view—in applications of poor architectural quality, causing tremendous costs for maintenance and further evolution [2]. The lack of standards and the broad diversity of design methods have led to the fact that no efficient concept for reuse of solutions and designs has been established yet. To support systematic UI design and development, a process incorporating a disciplined form of reuse is needed. The lack of reuse methods in UI design motivated us to explore patterns as core constituents for a novel reuse approach. Patterns help in the reuse of well-known and proven design solutions, and play a significant role throughout the whole UI development process.

Historically, best practices reusability in 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 [3], and the design of interactive systems can be facilitated by applying best design practices. In current practice, tools for capturing and disseminating

A. Seffah
Concordia University, HCSE Lab., Montreal, QC, Canada
e-mail: seffah@cse.concordia.ca

M. Taleb (✉)
École de technologie supérieure (ETS),
GELOG Lab. of Software Engineering and IT,
University of Quebec, Montreal, QC, Canada
e-mail: mohamed.taleb.1@ens.etsmtl.ca

design knowledge include guidelines, claims and patterns. Guidelines concentrate most often on the physical design attributes of the user interface, and examples are the Macintosh Human Interface Guidelines [4] and the Java Look and Feel Design Guidelines [5]. Claims [1] are 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. 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 [6,7], while claims are too grounded in specific scenarios and examples, limiting their generality.

During the last 5 years, the HCI investigated patterns, defined in Sect. 3, as a mean to capture and communicate the best practices of user interface design. A pattern is a named, reusable solution to a recurrent problem in a particular context of use. Patterns include specific attributes, and can be combined together in related pattern languages, resulting in both a lingua franca for design [8] and facilitating pattern-oriented design. They are applicable to different levels of abstraction, and are extremely useful for designers because they have the potential to drive the UI design process [9–11].

One important remark that needs to be made at the forefront of this paper is that patterns are a great source of interest not necessarily because they provide novel ideas to the software engineering community, but because of the way that they package already available design knowledge. This way of presenting information to software designers and developers allows the reuse of best practices, and avoids reinventing the “wheel” each time. In addition, HCI patterns are a great way of incorporating usability best practices into software development. In light of this, pattern use has not just gained popularity amongst software engineers, but is of great interest to usability engineers and specialists who are concerned with the construction of usable systems.

In this paper, we will review the evolution of pattern usages in UI design. The use of patterns has evolved dramatically over the last few years—from individual pattern use for solving a specific design problem, to pattern-oriented design, which guides designers in building a conceptual design by leveraging relationships between patterns. One of our underlying goals is to investigate how patterns can be used, not only to foster the reuse of proven and valid design solutions, but also as a central artefact in the process of deriving a design from user experiences and requirements. We will present our investigations on pattern-based design, and discuss how user experiences can be incorporated in the pattern selection process through the use of behavioural variables, pattern attributes and associated relationships.

2 Related works

2.1 Original ideas about patterns in design

The architect Christopher Alexander first introduced the concept of design patterns in the late 1970s. In his two books, *A Pattern Language* [12] and *A Timeless Way of Building* [13], he discusses the capture and use of design knowledge in the format of patterns, and presents a large collections of pattern examples to help architects and engineers with the design of buildings, towns, and other urban entities. To illustrate, Alexander proposes an architectural pattern called *Wings of Light* [12], where the problem is:

“Modern buildings are often shaped with no concern for natural light—they depend almost entirely on artificial light. But, buildings which displace natural light as the major source of illumination are not fit places to spend the day.”

Amongst other information such as design rationale, examples, and links to related patterns, the solution statement is:

“Arrange each building so that it breaks down into wings which correspond, approximately, to the most important natural social groups within the building. Make each wing long and as narrow as you can—never more than 25 feet wide.”

All of Alexander’s patterns address recurrent problems that designers face by providing a possible solution within a specific context. They follow a similar structure, and the presented information is organized into pattern attributes, such as problem and design rationale. Most noteworthy, the presented solution statement is abstract enough to capture only invariant properties of good design. The specific pattern implementation is dependent on the design details and the designer’s creativity [14]. In the example above, there is no mention of specific details such as the corresponding positions of wings to one another, or even the number of wings. These implementation details are left to the designer, allowing for different instances of the same pattern solution.

In addition, Alexander [12] recognized that the design and construction of buildings required all stakeholders to make use of a common language for facilitating the implementation of the project from its very beginnings to completion. If organized properly, patterns could achieve this for all the participants of a design project, acting as a communication tool for design.

The idea of leveraging these inherent properties of patterns has been quite influential in a variety of domains in the last decade. The use of original Alexandrian-type patterns as a

design tool has influenced areas as diverse as software development, organizational processes, and teaching. In software engineering, patterns exist at different levels of abstraction, and are process or product related. They can be associated to either the organization as a whole, the process of software development, the people involved, or the product being engineered. Alexander's pattern framework has been applied extensively to object-oriented programming and has inspired a different way of thinking in which design knowledge is captured and reused effectively. Alexander's influence is apparent in Gamma et al.'s book [15], "Design Patterns: Elements of Reusable Object-Oriented Software" which inspired the software engineering community to take a closer look at the concept of patterns as a problem-solving discipline for object-oriented design. Gamma et al. [15] have documented 23 design patterns in their book, one example being the Observer Pattern. Like all other patterns, the observer pattern is described in a specific format, with consistent attributes. A short description of this pattern is given in Table 1. For comparative purposes, Table 1 also illustrates an organizational pattern called Review the Architecture, by Coplien [16]. Organizational patterns are also documented in a specific format, with consistent attributes. Although these attributes may differ from those used to describe software design patterns, the principle is similar.

Alexander argues that traditional architectural design practices fail to create products that meet the real needs of the user, and are ultimately inadequate in improving the human condition. His patterns were introduced in a hierarchical collection with the purpose of making buildings and urban entities more usable and pleasing for their inhabitants. Interestingly enough, this very same idea can be extrapolated to HCI design, where the primary goal is to make interactive systems that are usable and pleasing to users. This will be discussed further in the next section.

2.2 HCI Patterns: definitions, examples and benefits

Within this paper, we will use the term user interface patterns, user interface design patterns and interaction design patterns interchangeably. The same for interaction design, user interface design and interactive systems design. Pattern has different definitions sometimes contradictory. From the most generic to more HCI domain dependant, a pattern is:

- Form, template, or model or, more abstractly, a set of rules which can be used to make or to generate things or parts of a thing.
- A general repeatable solution to a commonly occurring problem.
- A three-part rule that expresses a relation between a certain context, a problem, and a solution" [13].
- An invariant solution to address a recurrent design problem within a specific context [14].
- A general repeatable solution to a commonly occurring usability problem in interface design or interaction design.
- A solution to a usability problem which occurs in different contexts of use.
- A successful HCI design solution among HCI professionals that provides best practices for HCI design to anyone involved in the design, development, evaluation, or use of interactive systems [9].
- A structured description of an invariant solution to a recurrent problem in context, reflecting Alexander's problem-oriented approach in software engineering and HCI [17].

In essence, patterns give an invariant solution to a problem and are abstract enough to draw on the common elements that hold between all instances of the resulting solution. What is notable about design patterns is that they are both concrete and abstract at the same time. They are concrete enough to provide sound solutions to design problems, which

Table 1 Example of a design and organizational pattern

Design pattern [15]	Organizational pattern [16]
Name: Observer	Name: Review the Architecture
Intent: Define a one-to-many dependency between objects. When one object changes state, all its dependents are notified	Problem: Blind spots occur in the architecture and design
Applicability: (1) A change to one object requires changing other unknown objects, (2) object should be able to notify other objects, but you don't want them to be tightly coupled	Context: A software artifact whose quality is to be assessed for improvement
Participants: Classes are Subject, Observer, Concrete_Subject, and Concrete_Observer	Forces: (1) A shared architectural vision is important, (2) even low-level design and implementation decisions matter, (3) individual architects and designers can develop tunnel vision
Consequences: (1) Abstract for broadcast communication, (2) support for broadcast communication, (3) unexpected updates	Solution: All architects should review all architectural decisions. Architects should review each other's code. The reviews should be frequent and informal early in the project
Related patterns: Mediator, Singleton	Resulting context: The intent of this pattern is to increase coupling between those with a stake in the architecture and implementation, which solves the stated problem indirectly
	Related patterns: Mercenary Analyst, Code Ownership

can be put immediately into practice. On the other hand, they are abstract enough to be applied to different situations.

HCI focuses on the design of usable systems, and HCI patterns are but one of a handful of design tools that provide a means to abstract and reuse the essential details of successful and usable design solutions. Prior to discussing patterns in detail, it is important to review guidelines and claims described in Sect. 2.3, two other tools that have influenced and promoted the reuse of design knowledge in HCI.

Above all, patterns are problem oriented, yet not toolkit specific. In addition, they are more concrete and easier to use for novice designers, context oriented, and promote reusability. Overall, patterns have a number of benefits:

- They are a relatively intuitive means to document design knowledge and best practices
- They are straightforward and readable for designers, developers and other stakeholders, and can therefore be used for communication purposes
- They come from experiments on good know-how and were not created artificially
- They represent design knowledge from different views, including social and organizational aspects, conceptual and detailed design
- They capture essential principles of good design by telling the designer what to do and why, but are generic enough to allow for different implementations

This last property is an especially discriminating characteristic of patterns, allowing them to give rise to different implementations of the same design solution. Different implementations are necessary to support variations in design look and feel, platform preference and usage context. Figure 1 illustrates how the Quick Access pattern, used to logically group the most frequently used pages on a Website, can be implemented on three different platforms. For a Web browser on a desktop, the Quick Access pattern is implemented as an index browsing toolbar; for a PDA, as a combo box; and for a mobile phone, as a selection [18].



Fig. 1 Quick access pattern

Another example is Overview and Detail (Table 2), a pattern for visualization environments. This pattern can be implemented differently by the designer, depending on variations in data and usage context. To illustrate, Windows Explorer and Google Maps demonstrate two different implementations. In Windows Explorer, the user is provided with two views—one that presents a hierarchical overview of folders, and the other, the contents of the selected folder. In Google Maps, the user is also provided with two views of the data—an orienting view of the selected area presented as a corner map, and a detailed view of the same geographic location.

In addition to the benefits described above, two cardinal properties of patterns have made their use increasingly valuable for designers. First, patterns include user-centered values within their rationale. Second, the concept of patterns and their associated pattern languages are generative, and can therefore support the development of complete design. The remainder of this paper will look at how these two properties have allowed patterns to evolve from a simple compilation of “best practices” to a powerful tool for designers, to be used as building blocks in a user-centered design process.

2.3 From guidelines and claims to patterns

In the 1990s, design guidelines became an increasingly popular way to disseminate usability knowledge and ensure a degree of consistency across applications [4, 19] and within organizations [20–22]. 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 4. To date, guidelines have yet to realize their full potential and have had little impact on the design of user interface software [23, 24]. Apart from not adequately addressing concerns facing designers, such as which guidelines should be used under what circumstances [25], studies have shown that interface guidelines suffer from being too abstract to be applied directly [6, 7]. 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 [1] 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 rationale. 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

Table 2 HCI pattern for visualization environments

Title	Overview and detail
Context	The dataset is large, too large for all the details to fit in a single view, and there is a need to view details about subsets of data items. The data can be viewed at one or more levels of abstraction e.g. directories and files within a directory, aggregated document content and detailed document content, etc. Alternatively the dataset may be large and continuous but only a subset can be viewed at any one time e.g. map data
Problem	How to display the entire contents of a large dataset at once, allow users to explore the dataset, and at the same time show details about subsets of items
Solution	Show an overview of the entire dataset together with some visual indication as to which part of the dataset is currently being viewed. Show details about subsets of items in a separate view The overview can be a scaled version of the main view, i.e. a spatial zoom, or some other representation, i.e. a semantic zoom. Since the overview tends to display a higher number of data items than any more detailed view it is necessary to use simple glyphs that minimize clutter, maximize use of screen space and portrait the data attributes most relevant to the task
Examples	Windows Explorer Google Maps
Other attributes	Forces, related patterns, design rationale

Table 3 An example of such a reused claim for a safety-critical application

Reused claim: safety-critical application
Claim ID: Rare event monitor
Target artifact: User interface for a chemical analysis instrument control system
Description: 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
Upside: Automatic detection of dangerous events relieves the user of constant monitoring; automatic detection and warning gives the user time to analyze the problem
Downside: Issuing too many warnings may lead the user to ignore critical events; automated monitoring may lead to user overconfidence in the automated system and decrease their situation awareness
Scenario: 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



generality because 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 3.

Patterns (see Table 4), the main focus of this paper, have been of recent interest to HCI practitioners and user interface designers. 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 pre-defined 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 because 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

Table 4 Guideline versus pattern for the toolbar

Guideline	Pattern
<ul style="list-style-type: none"> ○ A <i>toolbar</i> is a collection of frequently used commands or options that appear as a row of toolbar buttons ○ 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 ○ Toolbars typically contain buttons, but you can provide other components (such as text fields and combo boxes) as well ○ 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>Pattern name: Convenient toolbar 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 <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>
	

Source: Java Look and Feel Design Guidelines:

<http://java.sun.com/products/jlf/ed2/book/>

applied. Since patterns are context-oriented, the solution is related to a specific user activity. Table 4 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” [1]. 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.

2.4 HCI pattern languages

A number of pattern languages have been suggested in HCI. For example, Van Duyne's “The Design of Sites” [26], Welie's Interaction Design Patterns [27], and Tidwell's UI

Patterns and Techniques [28] play an important role and wield significant influence in the HCI community. In addition, specific languages such as Laakso's User Interface Design Patterns [29] and the UPADE Web Language [18,30,31] have been proposed.

Patterns are organized into pattern languages. Just as words must have grammatical and semantic relationships to 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.

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 (Tables 3 and 4)—with the common attributes context, problem, solution, forces, related patterns, and examples. Secondly, the language must logically group patterns.

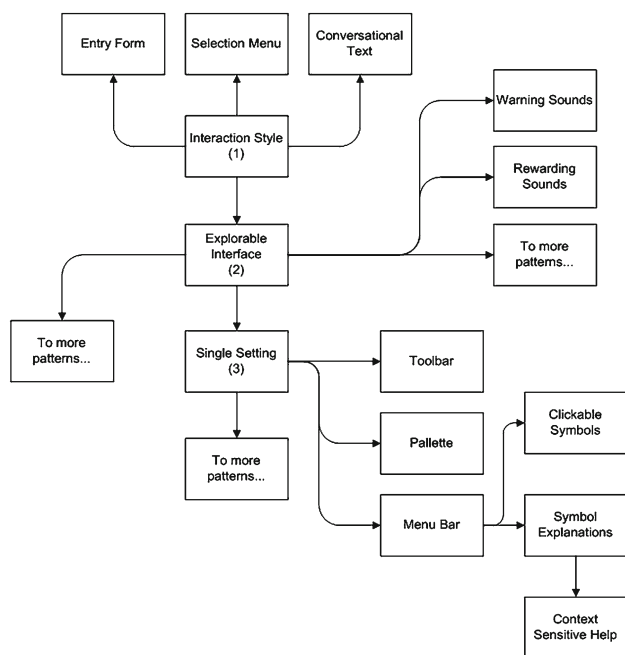


Fig. 2 The experiences pattern language

Tidwell [32] 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), developed by Coram and Lee [33], 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. Thirdly, pattern interrelationships should be described. In Experiences, 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.

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 meta-pattern interaction style (denoted by “(1)” in Fig. 2) is the first pattern that leads the designer along the major paths through the language. The design advice [ref: Experiences] for this pattern includes studying the user and environment, working with the user to determine what interaction style is best, and keep-

ing the interface simple and consistent. This pattern is connected to four other patterns as indicated by arrows in Fig. 2 (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.

2.5 Patterns 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 [34]. 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 [35].

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 [8] for design, which can be read and understood by all. Erickson [8] 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 which can guide software designers through the design process. However, there exists no commonly agreed upon UI design process that employs pattern

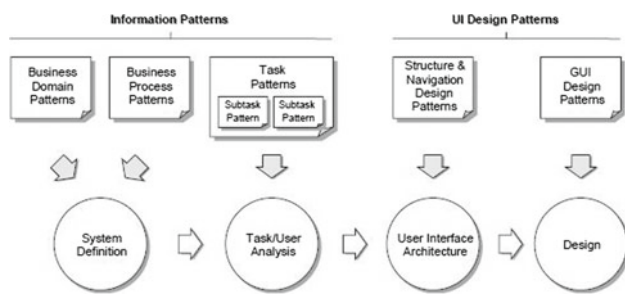


Fig. 3 The pattern-supported approach framework [10]

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), HCI patterns are used at various levels to solve problems relating to business domains and processes, tasks, structure and navigation, and GUI design [10]. The main idea that can be drawn from PSA is that HCI patterns can be documented identified and instantiated according to different parts 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.

Van Duyne et al. [26] 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 groups generally indicate the order in which they should be used in the design process (Table 5). 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.

3 Pattern-oriented design

Javahery and Seffah [11] have proposed a 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 UPADE¹ Web language [30, 18, 31]. 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.

POD has two features. First, it provides a framework for guiding designers through stepwise design suggestions. At each predefined design step, designers are given a set of patterns which 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. Second, 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 sub-categorized into Page manager and Information container patterns (see Fig. 4 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.

In POD, designers first should follow a POD model. We have published literature on a preliminary version of this model [11]. As part of this paper, 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,

¹ Usability pattern assisted design environment (UPADE) is a Web language and design environment developed by Concordia University's Human-Centered Software Engineering Group.

Table 5 Pattern groups ordered according to a Web design process [26]

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
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

we define four steps that designers should follow: (1) Defining the architecture of the site with architectural patterns, (2) establishing the overall structure of each page with page manager patterns, (3) identifying content-related elements for each page with information container patterns, and (4) organizing the interaction with navigation support patterns. Landay and Myers [36] and Welie [37] also propose to organize their Web pattern languages according to both the design process and UI structuring elements (such as navigation, page layout and basic dialog style).

The first step of the POD process begins with the description of the architecture of the entire Web site, and application of architectural patterns. Four basic patterns can be used to organize the content of a complex Web site. These patterns are the Sequence, Hierarchical, Grid and Composite patterns. The simplest architectural pattern is the Sequence pattern which organizes Web application content as a sequence, or a linear narrative. The Hierarchical pattern is a tree-based hierarchy, and is one of the best ways to organize complex bodies of Web information. Hierarchical organization schemes are particularly well suited to organizing a complete Web site. Finally, the Grid pattern should be used when topics and contents are fairly correlated with each other, and there is no particular hierarchy of importance. Procedural manuals, lists of university courses or medical case descriptions are often

best organized using Grid patterns. For larger and more complex Websites, a combination of these basic patterns is often required, referred to as the Composite pattern. Figure 6 illustrates the Composite pattern. Among the many relationships that exist between these three basic patterns, we note that the Composite pattern is super-ordinate to the Sequence, Grid and Hierarchical patterns.

Secondly, designers should exploit relationships between patterns. We have described five types of relationships between the UPAD patterns, published in [18,38]. The same relationships can easily be applied to other pattern libraries. This multi-criterion classification is based on the original set of relationships [26,39,40] used to classify the patterns proposed in Gamma et al. [15]. The relationships are used to compose a UI design, allowing designers to make suppositions such as: "For some problem P, if we apply Pattern X, then Patterns Y and Z apply as sub-ordinates, but pattern S cannot apply since it is a competitor." The relationships are:

1. *Similar* is a relationship, which applies to the same category of patterns.
2. *Competitor* is a relationship that applies to two patterns of the same patterns category.
3. *Super-ordinate* is the basic relationship to compose several patterns of different categories.

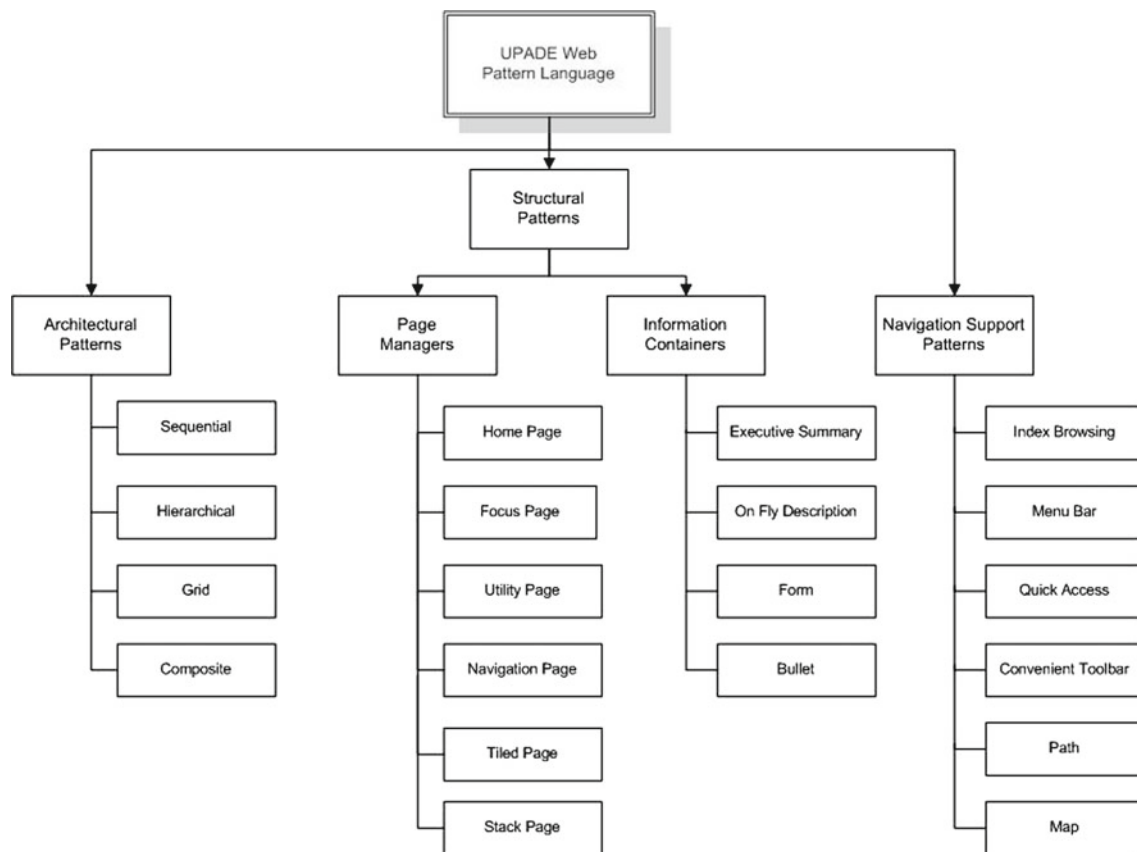


Fig. 4 An overview of the UPADE pattern language

4. *Sub-ordinate* (X, Y) if and only if X is embeddable in Y. Y is also called super-ordinate of X. This relationship is important in the mapping process of POD from a platform to another one.
5. *Neighboring* (X, Y) if X and Y belong to the same pattern category (family).

Within the scope of the development of Web-based applications using the UPADE language, POD allows for the exploitation of 48 pattern relationships, allowing even novice developers to use the underlying best practices to iterate through concrete and effective design solutions. As described in our pattern model, each pattern contains a list of related patterns. For example, the Stack page pattern would contain the following information about related patterns: (1) Super-ordinate: Sequential, Hierarchical, Grid, Composite (Fig. 5). (2) Sub-ordinate: Executive Summary, On Fly Description, Browsing Index. (3) Competitor: Focus Page, Tiled Page. Let us illustrate how pattern composition can be applied to our Website design. We will use the POD model in combination with the patterns selected from [18].

During the second step, the designer applies structural patterns to establish a consistent physical and logical screen layout for each page that was defined in the previous step.

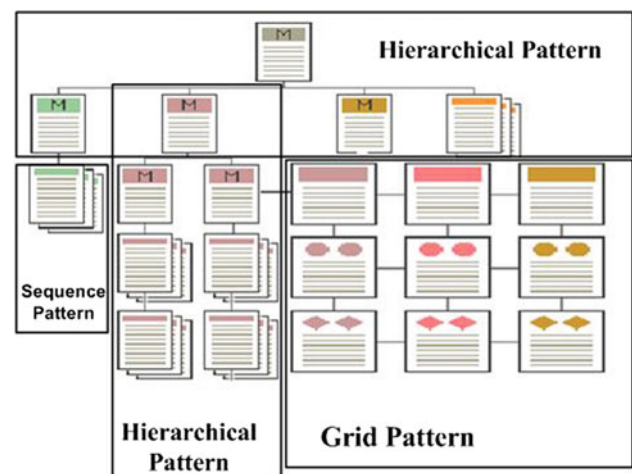


Fig. 5 The Composite pattern

This step involves applying page manager patterns, which are a type of structural pattern (Table 6). Different relationships exist between these patterns, and even between these patterns and those used in the previous design step. As an example, all the Structural patterns are subordinate to the Architectural patterns from the last step. In addition, to further illustrate some relationships, Tiled page and Stack page patterns are

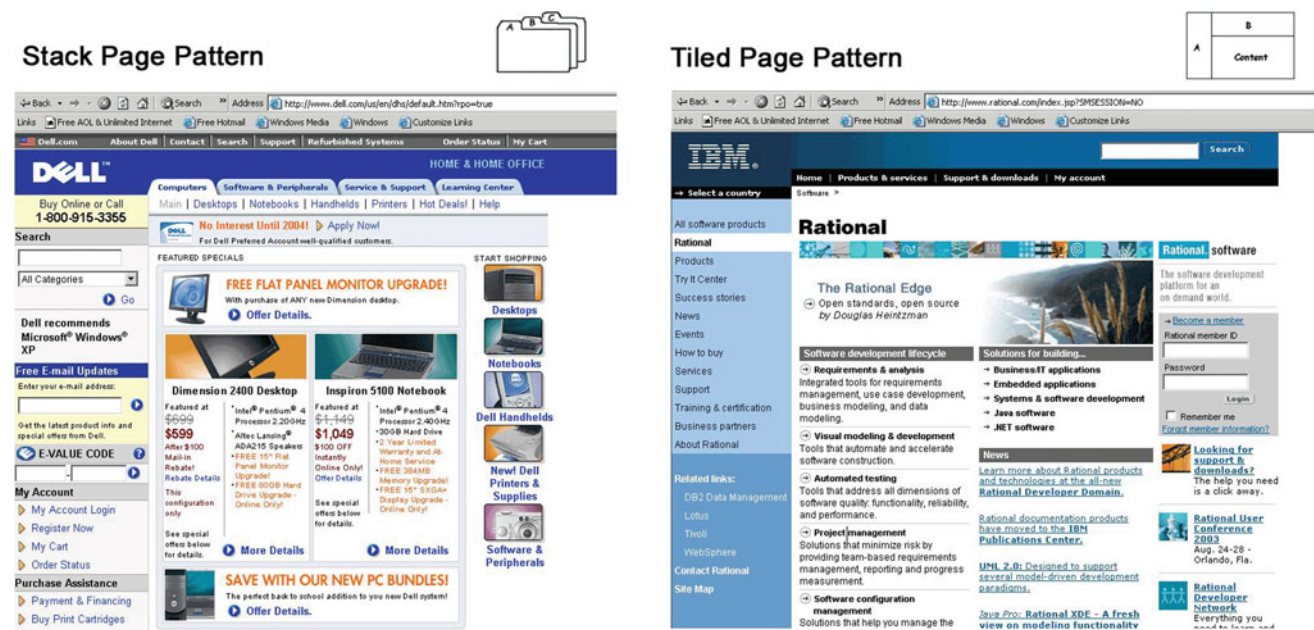


Fig. 6 Comparison of Stack and Tiled Page patterns

competitors (Fig. 6). This means that if you choose the Tiled page pattern as a basic model for your home page, you cannot use the Stack page pattern for any of the subsequent pages. Such knowledge can be critical for pattern users because if it is not taken into consideration during design, it can compromise the benefits of the pattern.

The third step of the POD process involves employing information container patterns, the second type of structural patterns, to quickly “plug in” an information segment for each page. Long before the Web was invented, authors of technical documents discovered that users appreciate short segments of information. Such design practices should be embedded in the design process and presented to the designer. For example, for users, how long does it take to determine if a large document contains relevant information? This question is a critical design issue, and the Executive summary pattern (Table 6), which provides an information preview, may be an appropriate design solution.

The fourth and final step consists of building the navigation support. It is possible to consider navigation elements earlier in conjunction with other patterns. Navigation support patterns suggest different models for navigating between information segments and pages. To illustrate an example of existing relationships, the Index browsing and Dynamic Path patterns (Table 6) are considered neighbouring since they belong to the same design step. Although they are both used for navigation support, they are not used to solve the same usability problem and are applied in different contexts. Dynamic Path is used to navigate between pages in an already-taken path, and gives the user a sense of safety and control. Index browsing is generally used to navigate

amongst important content pages, and allows the user to reach these pages safely.

Although all of the patterns can be applied independently, one of the main strengths of the POD approach is that developers can exploit pattern relationships and apply this knowledge to their design solutions. As an example, the Executive Summary pattern, combined with the Index browsing patterns from Navigation Support, allows users to preview information about a certain topic before spending time to download, browse and read different pages (Fig. 7). Executive Summary is weighted as a highly recommended subordinate pattern when pattern users try to use the Index browsing pattern.

Knowledge about context-oriented relationships, as described above, can be very useful to pattern users. They can be a guide in choosing the best solution for a specific user problem based on a particular context. Novice designers and software developers who are unfamiliar with user-centered design and usability engineering can especially benefit from such a systematic design process.

4 Patterns and model-driven design

In previous sections, we discussed the evolution of pattern use from single pattern use to pattern languages with relationships, and finally, to design processes driven solely by patterns. Recently, some authors have suggested using patterns to supplement existing design methods. In this section, we summarize how patterns have been used within two popular UI design approaches: model-driven and component-driven design.

Table 6 Pattern examples used in the POD process

Pattern type	Pattern name	Description
Structural patterns:	Tiled page	Structures and presents content to the user from more general to specific by dividing the page into several surfaces
Page managers	Stack page	Groups content into categories which have no obvious hierarchy; this is done by designing several surfaces stacked together and labelling them appropriately
Structural patterns:	Executive summary	Provides an information preview or summary for a certain topic of choice
Information containers	On fly description	Provides the user with a short description of the object when the mouse hovers over it
Navigation	Dynamic path	Indicates the user's entire path starting from when the Web application was initially accessed, and is similar to "breadcrumbs" in other pattern languages
Support patterns	Index browsing	Allows the user to easily and promptly navigate amongst important content pages, and is located consistently throughout the Website

**Fig. 7** The Index browsing and Executive Summary pattern

Central to all model-based approaches is that relevant aspects of a user interface design are represented using declarative models. In a model-based approach, the UI design is the process of creating and refining the user interface models at different levels of abstraction [41], with a focus on finding mappings between the various models [42]. Eventually, user interfaces are generated automatically or semi-automatically with the designer's interventions and decisions, from the model descriptions. In model-based UI design, early design decisions are typically made during the creation of the envisioned task model. Additionally, models for capturing user characteristics and business objects are developed. Based on these rather abstract models, a dialog, a presentation and a layout model are derived to reveal some implementation details of the user interface.

Sinnig, Molina, Trætterberg [43–45] have demonstrated how patterns can be used in conjunction with models to support the UI design process. For example, Sinnig et al. [43] describes a framework whereby patterns are used to describe

model fragments. The approach aims to use patterns as building blocks in order to create these various models. An underlying reason for the use of patterns in this framework is to increase reuse of already modeled solutions. In order to foster re-use in different contexts of use, patterns are introduced as abstractions, which must be instantiated. In particular, it is demonstrated how different kinds of patterns can be used as building blocks for the creation of UI models.

One underlying principle in pattern use for model-based design is the formalization of patterns. Molina and Trætterberg [44] also recognized that the mostly informal description of today's patterns is not suitable for processing them by tools. Within his Just-UI framework, he proposes a formalized set of patterns for the presentation model. Another prerequisite for effectively using patterns is to keep their entailed design solutions generic enough to be applicable in different contexts of use. Before the design solution stated in the pattern is really tangible and applicable, it must be adapted to the current context of use and domain. In Sinnig et al. [43], it is proposed to attribute patterns with variables, which can act as placeholders for context-dependent design details. During the process of pattern application, the variables must be bound to concrete values representing the surrounding context.

Thus far, we have concentrated on patterns as design building blocks, providing solutions to common design problems at a conceptual level. Patterns, in general, are of rather conceptual nature and abstract from concrete implementation-language issues. This generic nature of patterns allows the

designer to think “outside” the toolkit at hand [32]. However, when it comes down to the implementation, the previously conceptually designed aspects for the application must be carried out using a particular programming environment and framework. At this point, software components can supplement patterns, by providing a concrete implementation of patterns at the code level. A software component can be defined as a unit of composition with a clearly defined interface, enabling independent deployment and third-party reuse. Components are a suitable means to implement patterns for a particular platform. Once an appropriate pattern has been chosen and instantiated, a suitable component can be picked to implement the particular pattern instance. In the same way as patterns, components can also be customized and adapted to different contexts of use.

To accomplish this, Sinnig et al. [46] have suggested a UI development process that incorporates both patterns and components. The pattern- and component-based (PCB) process for systematic UI development incorporates patterns and components as its core constituents. Patterns are used as building blocks to establish conceptual UI design models, whereas components are introduced as a means to implement particular pattern instances. The starting point is the selection and instantiation of applicable patterns (phase 1 in Fig. 8). Like POD and PSA, the PCB process begins after the analysis phase and requirements specification. It is assumed that the problem domain has already been precisely specified. Next the various pattern instances are associated to different declarative UI models (phase 2 in Fig. 8) and used as building blocks to form the various models. Then components are picked in order to implement the UI according to design information entailed in the UI models (phase 3 in Fig. 8). In particular, since the models are mainly formed and established by patterns, relevant components are chosen to implement the various patterns. Finally the various components are adapted to the context of use and are linked with each other and deployed to assemble the UI of the host application.

5 Linking patterns to user experiences

From a software engineering perspective, advances in pattern use have come a long way. The past 5 years have seen an evolution in patterns—from tackling a specific design problem, to composing comprehensive designs. Furthermore, linking patterns to a design approach, and the incorporation of patterns in existing design frameworks, has strengthened their use. Most recently, researchers have used patterns in conjunction with UI models and components, bridging the gap between design and system implementation. Differentiating them significantly from guidelines and claims, patterns are no longer simply “another” technique from the UI

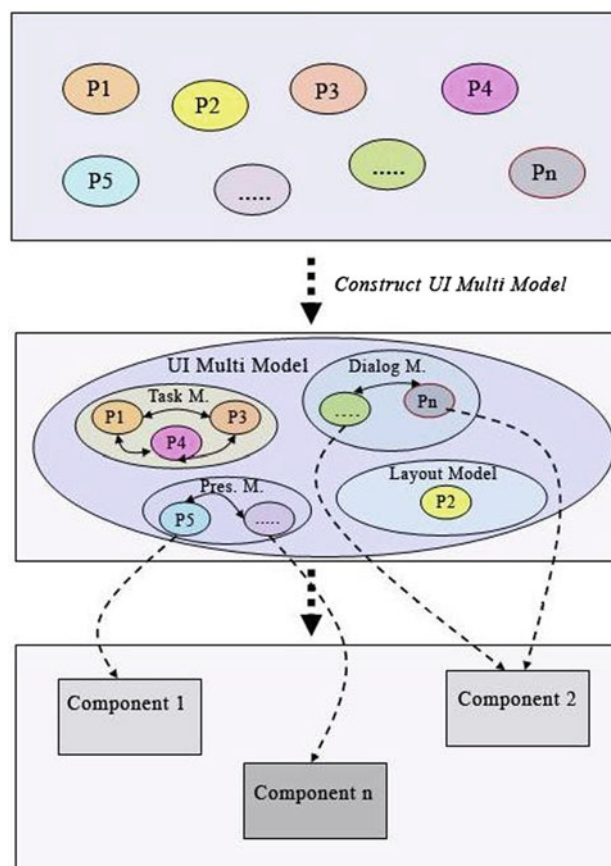


Fig. 8 The pattern and component-based UI development process

designer’s toolbox, but are emerging as possible core constituents of the UI design process.

However, saying that, one important aspect of patterns—most relevant for HCI and UI design—has been grossly neglected. We cannot forget that HCI focuses on the design of usable systems, with a fundamental mission being the incorporation of psychology and other behavioural sciences to bear upon design. Although pattern use has advanced significantly from a software-implementation perspective, it has a long way to go to fulfill its original intent of providing solutions for usability-specific problems in HCI. To achieve such a feat, patterns should somehow be associated with users and their experiences.

Current pattern languages include little information about users. When they do, they are hidden amongst paragraphs of textual information contained within pattern descriptions. For example, the “Redundant Encoding” pattern indicates amongst its context that users may have visual deficiencies such as color blindness. Another example is the “Wizard” pattern [47] that gives guidance to users. It is important for designers to know what kind of added value patterns bring, especially to their users and in terms of usability. This kind of information should not simply be hidden in pattern

descriptions, but should be part of the elements that make up a pattern.

Welie [47] and Traetteberg [48] discuss how patterns are presented from a designer's perspective, with little consideration to user problems and usability principles. They propose presenting patterns from an end-user's perspective, with a focus on usability as the essential design quality including information about user problems in their UI design patterns.

Pattern-based designs should be considered as design blocks that apply in particular situations, derived from an understanding of user experiences. User experiences is an umbrella term referring to a collection of information that covers user behaviors (observed when the user is in action), expectations, and perceptions—influenced by user and application characteristics. User characteristics include knowledge, experience, personality and demographics. Application characteristics include domain, content, language, visual design and interaction type. Other context of use information, such as technical, social and organizational characteristics, may also have an influence on user experiences.

Ethnographic and empirical techniques are generally used to collect relevant user data to describe user experiences, ranging from field studies, observations, and task-based evaluations. Different user-centered techniques and artifacts use this data to represent user experiences, mostly in narrative form: Character maps [49], personas [50], task analysis and scenario-based approaches [51] are the most popular.

Among the various techniques to capture user experiences, persona is a descriptive model of the user, encompassing information such as user characteristics, goals and needs. In current practice, they are primarily being used as a communication tool. Cooper [50] describes a seven-step process to persona creation with intermediate steps that include identifying significant behavioral patterns and designating persona types. Pruitt and Grudin's [52] approach makes persona creation more rigorous with links to real data. We propose to extend these ideas by placing personas within a design framework linked to collected user data and pattern-based design solutions. By representing personas in a more discrete manner and using behavioral variables, we can correlate user information to patterns. In this way, information contained within personas will not simply "inspire" members of the development team to design interactive systems accordingly, but play a part in providing concrete design solutions.

The persona to pattern framework (Fig. 9) supports pattern selection and combination to create a conceptual design from extended personas. The starting point consists of creating a set of personas to describe the user experiences of an existing or envisioned system. Similar to Pruitt and Grudin [52], we believe that personas should be based on empirical evidence with links to real data. We have employed heuristic evaluation

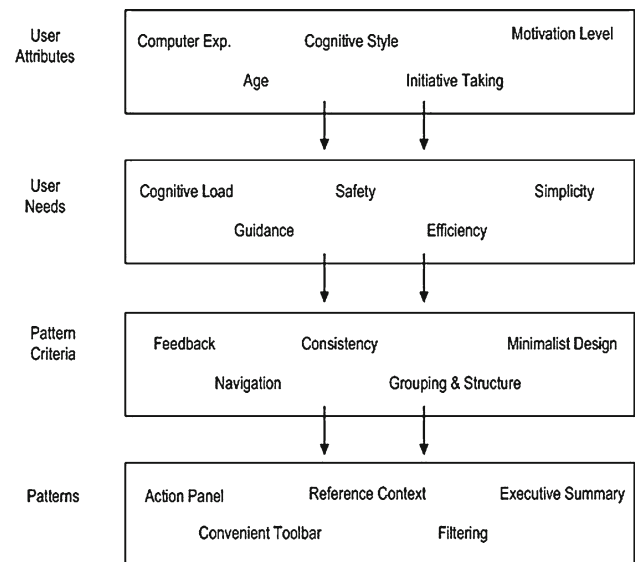


Fig. 9 The persona to pattern framework

and psychometric assessment to enhance our persona data. The observations made with users during these usability evaluation steps were used to: (1) Further enhance the set of original personas and determine precise interaction behavior in light of user goals and tasks, (2) determine usability problems with current or similar applications. Both these points are essential for selecting and composing appropriate patterns as part of the design process.

In order to increase persona effectiveness as a technique, we developed our personas iteratively using ethnography, domain analysis and empirical evidence. They were extended to include information about interaction behavior, user tasks and goals. The starting point in such a process is collecting user experiences with an existing or an envisioned system, iteratively via empirical studies. The original set of users is then grouped into an initial set of personas that can be refined using clustering techniques. The mapping step consists of logically linking a set of design patterns to a set of personas. Using relevant relationships between patterns, the identified patterns are then selected and composed to create a pattern-oriented design. At the core of the derivation process are behavioral and P variables, their relationships and associated algorithms, the four milestones (clustering, mapping, pattern selection and composition), and the Persona to Pattern Mapper support tool.

The overall purpose of this project is to find ways to guide designers in choosing patterns based on user attributes. The first step is to link user attributes and variables (such as age, cognitive style, etc.) with user needs. The assumption is that certain types of users have differing needs relative to the design of a particular system, from a usability point of view. These user needs will then be mapped to more concrete pattern criteria, which will help designers choose patterns for their designs. The overall idea is as follows:

6 The needs for patterns representation and semantics

The look at patterns in general as artifacts that have three main milestones, organized from a user perspective (Fig. 10).

On the pattern user side, we can say that patterns are harvested and represented with the main goal of being delivered to other users who implement them as solutions. A delivery paradigm is essential in the pattern approach because it indicates that patterns arrived effectively to potential users; a knowledge dissemination view. This means that patterns should be represented in a way that software developers can learn, master and apply easily and effectively in their context. This implementation highlights the main role of patterns, promoting effective reuse. If patterns were harvested and written down just for the sake of archiving them then we have missed on the great benefits of patterns.

On the pattern writer side, the discovery of a pattern is only the beginning. Harvesting is a carefully selected metaphor that indicates the hard work associated with patterns. By observing existing artifacts and problems that have been solved successfully, we can detect a repeated structure or behavior that is worth recording. By asserting its importance, we can write down the essential components and—if possible—analyze them. An expert can provide insight as to why this combination is good or why it works well and in what context. Finally guidance of how to reuse this solution can be added to assist in modifying and reapplying the solution.

Representation of patterns can be seen as a vehicle—a medium or an infrastructure to bridge the gap between the two main activities; delivery and discovery. This representation is essentially about how to format the solution in a way that allows it to mature from its solution-format into a pattern. In essence, a pattern is a solution alongside other information that supports it. The reason is that in order for a solution to be used by others, they have to be convinced that this is a good solution. Part of this comes by annotating pattern solution with expert analysis and comments, listing of some cases where the solution has been applied and the “success indicators”, and possibly some code examples. Bearing in mind

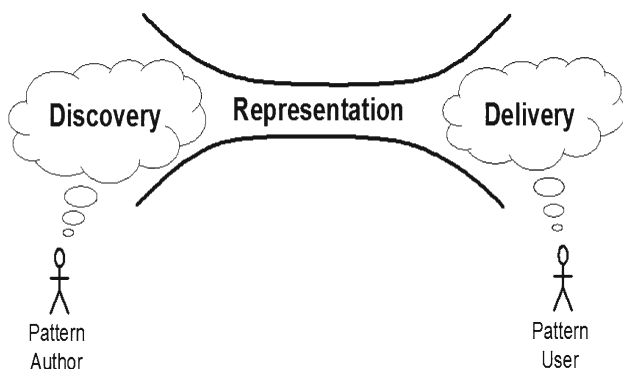


Fig. 10 Major milestones and users of patterns

that no two systems are exactly the same, and that every new software is a new adventure, patterns are typically annotated with important guidance on how to apply them in different contexts and situations. Some details are left out to allow the end user to rematerialize an abstract pattern back into a concrete solution that is adapted to the new design. Having decided on what to write, the sibling question would be how to best represent this information: through UML diagrams, simple diagrams, images, text, source code, or a combination of all of them.

The success of pattern approach depends on all those three milestones. As we discuss the potential benefits of applying patterns in design reuse, we cannot claim that patterns are “silver bullets”. Due to the inherently creative nature of design activities, the direct reusability of designs represents only a small portion of the total effort. It requires a considerable amount of experience and work to modify existing designs for reuse. Many design ideas can only be reused when abstracted and encapsulated in suitable formats. Despite the creative nature of their work, software designers still need to follow some structured process to help control their design activities and keep them within the available resources. Within this process, tools can help glue patterns together at higher design levels the same way we do with code idioms and programming language structures. For example, the Smalltalk Refactoring Browser, a tool for working with patterns at the source code level, assists developers using patterns in three ways [53]:

- Generate program elements (e.g. classes, hierarchies) for new instances of a pattern, taken from an extensible collection of “template” patterns.
- Integrate pattern occurrences with the rest of the program by binding program elements to a role in a pattern (e.g. indicating that an existing class plays a particular role in a pattern instance).
- Check whether occurrences of patterns still meet the invariants governing the patterns and repairing the program in case of problems.

7 Conclusion

Although patterns design building blocks having irrefutable benefits, their current use needs to be further structured and formalized within the HCI community. To begin with, a universally accepted taxonomy for pattern languages is still missing in HCI. Patterns deal with different levels of abstraction within design. Therefore, if languages are not structured logically, it can be confusing for designers trying to work with them. Some languages have suggested their own partial classifications to facilitate the use of patterns. For example, Welie [47] discusses a taxonomy based on the domain

of application—Web, GUI or Mobile UI design patterns. Tidwell [32] organizes her patterns according to different facets of UI design; categories include Content Organization, Navigation, Page Layout, and Actions/Commands.

In addition, pattern languages need to clearly define pattern relationships. Currently, pattern interrelationships are often incomplete and not context-oriented. This is, by far, the most serious drawback of current languages. For example, the Experiences language describes some pattern relationships, but is incomplete. Other languages mention “related patterns” in their descriptions, but do not define the precise nature of the relationship. This is a limitation since relationship definitions are an important factor in determining the circumstances under which a pattern is applicable, having an effect on the pattern’s context of use.

Furthermore, there are no standards for the documentation of patterns within each language. This can be viewed as a side effect of the problem but the solution already exists for this type of issues. The current pattern languages use the narrative text formats, with varying attributes that gives rise to misunderstandings, confusion, and desperation. This situation makes them hard for use, especially when it comes to combine patterns from different languages.

Finally, a language in computer science is described as having some sort of syntax and semantic. None of the current pattern languages follow this principle, and there are no universally applicable rules in the HCI community with regard to how to document patterns and structure pattern languages. This has implications with regard to how usable current pattern languages are for designers. Such issues are currently being explored and tackled by various HCI conference workshops every year.

References

1. Sutcliffe A G (2000) On the effective use and reuse of HCI knowledge. *ACM Trans Comput Hum Interact* 7(2):197–221
2. Gaedke M, Segor C, Gellersen H-W (2000) WCML: Paving the Way for Reuse in Object-Oriented Web Engineering. *SAC* (2) 2000:748–755
3. Myers BA, McDaniel RG, Kosbie DS (1993) Marquise: creating complete user interfaces by demonstration. *INTERCHI 1993*:293–300
4. Macintosh (1992) *Human Interface Guidelines*. Apple Computer Company, Cupertino
5. Sun Microsystems (2001) *Java Look and Feel Design Guidelines*. Addison-Wesley Professional, Reading
6. Tetzlaff L, Schwartz DR (1991) The use of guidelines in interface design. In: *Proceedings of CHI’91*, vol 91, pp 329–333
7. Thovtrup H, Nielsen J (1991) Assessing the usability of a user interface standard. In: *Proceedings of ACM CHI’91 Conference. Human Factors in Computing Systems*. New Orleans, LA, 28 April–2 May, pp 335–341
8. Erickson T (2000) *Lingua Franca for Design: sacred places and pattern languages*. In: *Proceedings of Designing Interactive Systems*. ACM Press, New York
9. Borchers J O (2001) *A Pattern approach to interaction design*. Wiley, New York
10. Granlund A, Lafrenière D (1999) PSA, A Pattern-supported approach to the user interface design process. Position paper for the UPA’99 Usability Professionals’ Association Conference, Scottsdale, AZ, 29 June–2 July 1999
11. Javahery H, Seffah A (2002) A model for usability pattern-oriented design. In: *Proceedings of TAMODIA 2002*, Bucharest, Romania, pp 104–110
12. Alexander C, Ishikawa S, Silverstein M (1977) *A pattern language: towns, buildings, and constructions*. Oxford University Press, New York
13. Alexander C (1979) *The timeless way of building*. Oxford University Press, New York
14. Dix AJ, Finlay JE, Abowd GD, Beale R (1998) *Human–computer interaction*, 2nd edn. Prentice-Hall, (Englewood Cliffs)
15. Gamma E, Helm R, Johnson R, Vlissides J (1995) *Design patterns: elements of reusable object-oriented software*. Addison-Wesley, Reading
16. Coplien JO, Schmidt DC (1995) *Pattern languages of program design*. Addison-Wesley, Reading
17. Dearden P, Finlay J (2006) Pattern languages in HCI: a critical review. *Human–computer interaction* 21(1):49–102
18. Taleb M, Javahery H, Seffah A (2006) Pattern-Oriented design composition and mapping for cross-platform Web applications. *The XIII International Workshop on Design, specification and verification of interactive systems*, Springer Verlag, Trinity College Dublin, Ireland
19. Microsoft (1995) *The Windows Interface Guidelines for Software Design*. Microsoft Press, Redmond
20. Billingsley P A (1995) Starting from scratch: building a usability program at Union Pacific Railroad. *Interactions* 2(4):27–30
21. Rosenzweig E (1996) Design guidelines for software products: a common look and feel or a fantasy? *Interactions*, 3(5):21–26. doi:10.1145/234757.234759
22. Weinschenk S, Yeo S C (1995) *Guidelines for enterprise-wide GUI design*. Wiley, New York
23. Gould JD, Boies SJ, Lewis C (1991) Making usable, useful, productivity-enhancing computer applications. *Commun ACM* 34(1):74–85. doi:10.1145/99977.99993
24. DSouza F, Bevan N (1990) The use of guidelines in menu interface design. In: *Proceedings of IFIP INTERACT ’90*. Cambridge, 27–31 August, pp 435–440
25. Henninger S, Haynes K, Reith MW (1995) A framework for developing experience-based usability guidelines. In: *Proceedings of the conference on Designing interactive systems: processes, practices, methods, & techniques*. Ann Arbor, pp 43–53. doi:10.1145/225434.225440
26. Van Duyne DK, Landay JA, Hong JI (2003) The design of sites: patterns, principles, and processes for crafting a customer-centered Web experience. Addison-Wesley, Reading
27. Welie MV (1999) *The Amsterdam Collection of Patterns in User Interface Design*. Available via DIALOG. <http://www.cs.vu.nl/~martijn/patterns/index.html> (Online)
28. Tidwell J (1997) *A pattern language for human–computer interface design*. Available via DIALOG
29. Laakso Sari A, (2003) *Collection of user interface design patterns* University of Helsinki, Dept. of Computer Science. <http://www.cs.helsinki.fi/u/salaakso/patterns/> (Online)
30. Engelberg D, Seffah A (2002) Design patterns for the navigation of large information architectures. *11th Annual Usability Professionals Association Conference*, Orlando, 8–12 July 2002
31. Taleb M, Seffah A, Abran A (2007) Pattern-oriented architecture for Web applications, 3rd International Conference on Web Information Systems and Technologies (WEBIST 2007), 3–6 March 2007, ISBN 978-972-8865-78-8, Barcelona, pp 117–121

32. Tidwell J (2004) UI Patterns and Techniques. <http://time-tripper.com/uipatterns/index.php> (Online)
33. Coram T, Lee J (1998) A pattern language for user interface design. Available via DIALOG. <http://www.maplefish.com/todd/papers/experiences> (Online)
34. Zimmerman J, Evenson S, Baumann K, Purgathofer P (2004) The relationship between design and HCI. Workshop of CHI Extended Abstracts 2004, pp 1741–1742
35. Myers BA, Rosson MB (1992) Survey on User Interface Programming. CHI 1992:195–202
36. Landay JA, Myers BA (2001) Sketching interfaces: toward more human interface design. *IEEE Comput* 34(3):56–64
37. Welie MV, Van der Veer Gerrit C (2003) Pattern languages in interaction design. In: INTERACT 2003
38. Javaheery H, Sinnig D, Seffah A, Forbrig P, Radhakrishnan T (2006) Pattern-based UI design: adding rigor with user and context variables. In: TAMODIA 2006, pp 97–108
39. Zimmer W (1995) Relationships between design patterns. Addison-Wesley Publishing, ACM Press, New York, pp 345–364
40. Yacoub S, Ammar H (2003) Composition of design patterns. Addison Wesley Professional, Hardcover, p 416, ISBN 0-201-77640-5.
41. De Silva P (2000) User interface declarative models and development environments: a survey. In: Proceedings of DSV-IS 2000. Springer, Berlin, pp 207–226
42. Vanderdonckt J, Furtado E, Furtado J, Limbourg Q (2003) Multi-Model and Multi-Level Development of User Interfaces. Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces. Wiley, London. pp 193–216
43. Sinnig D, Gaffar A, Reichart D, Forbrig P, Seffah A (2004) Patterns in model-based engineering. In: Proceedings of CADUI 2004 jointly organized with ACM-IUI 2004, Funchal, 13–16, pp 197 – 210
44. Molina P, Trätteberg H (2004) Analysis & design of model-based user interfaces. In: Proceedings of CADUI 2004, 13–16, Funchal, pp. 211–222
45. Trätteberg H (2002) Using user interface models in design. In: Proceedings of CADUI 02, France
46. Sinnig D, Javaheery H, Forbrig P, Seffah A (2005). Patterns and components for enhancing reusability and systematic UI development. In: Proceedings of HCI International, Las Vegas
47. Welie MV (2004) Patterns in interaction design. Available via DIALOG. <http://www.welie.com>
48. Trätteberg H (2004) Integrating dialog modeling and application development. In: Making model-based UI design practical: usable and open methods and tools, A Workshop at IUI 2004, Madeira
49. Verplank B, Fulton J, Black A, Moggridge B (1993) Observation and invention: Use of scenarios in interaction design. Handout for Tutorial, INTERCHI'93, Amsterdam, 1993
50. Cooper A (1999) The inmates are running the asylum: why high-tech products drive us crazy and how to restore the sanity, SAMS Publishing, Indianapolis
51. Carroll JM (2000) Scenario-based design of human-computer interactions. MIT Press publishing, Boston
52. Pruitt J, Grudin J (2003) Personas: practice and theory. In: Proceedings of the 2003 Conference on Designing for User Experiences DUX '03, ACM Press, New York, pp 1–15
53. Fowler M (1997) Analysis patterns, reusable objects models. Addison-Wesley, Reading