Patterns of HCI Design and Design of Patterns

Ahmed Seffah
Human-Centered Software Engineering Group
Department of Computer Science and Software Engineering
Concordia University, Montreal Canada
seffah@cse.concordia.ca

ABSTRACT
User interface design patterns also called HCI or interaction or usability patterns have been introduced first as a medium to capture and represent solutions to users' problems. Patterns have been used also as a medium for transferring the expertise of HCI designers and usability professionals to software engineers, who are usually unfamiliar with UI design and usability principles. Design patterns have been considered also as a *lingua franca* for crossing cultural and professional barriers between different stakeholders. Several HCI professionals have introduced their own pattern languages with specific terminology, classification and meanings. Patterns have also been presented as building reusable blocks at different levels of granularity, which can be combined to compose new interactive systems. Despite the obvious and acclaimed potential of these pattern-driven design approaches, patterns usage has not achieved the acceptance and widespread applicability envisaged by pattern pioneers such as Christopher Alexander. This paper provides an analysis of the facts about patterns usages, pattern languages and pattern-based design approaches. Some shortcomings in the presentation and application of HCI patterns are identified and discussed under the prevailing fallacies. Based on the analysis of how patterns have used so far, we draw some recommendations and future perspectives on what can be done to address the existing shortcomings. Making patterns more accessible, easily understandable, comparable and integratable in software and HCI design tools can promote HCI patterns to claim the usability, usefulness and importance originally envisaged for the pattern-oriented design approach.

Categories and Subject Descriptors: D.2.2 [Design Tools and Techniques], H.5.2 [User Interfaces]

General Terms: Design, Theory, Human Factors

Keywords: Design patterns, pattern-oriented design, human-computer interaction, design methods

1. FROM BUILDING TO SOFTWARE DESIGN PATTERNS

Among the early attempts to capture and use design knowledge in the format of patterns, the first major milestone is often attributed to the architect *Christopher Alexander,* in the late 1970s. In his two books, *A Pattern Language* (Alexander, 1977) and *A Timeless Way of Building,* 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* (Alexander, 1977), 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."*

According to Alexander, every pattern has three essential elements, 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. Generally, the problem describes when to apply the pattern. The solution refers to a design form or a design rule that can be applied to resolve the forces. Solution describes the elements that constitute a pattern, relationships among these elements, as well as responsibilities and collaboration.

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. In addition, (Alexander, 1977) 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.

In Notes (Alexander, 1964), 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.

The pattern concept was not well known until 1987 when patterns appeared again at OOPSLA, the object orientation conference in Orlando. There Kent Beck and Ward Cunningham (Beck and Cunningham, 1987) introduced pattern languages for object-oriented software construction in a seminal paper**.** Since then many papers and presentations have appeared, authored by renowned software design practitioners such as Grady Booch, Richard Helm, Erich Gamma, and Kent Beck. In 1993, the formation of (Hildside Group, 1993) by Beck, Cunningham, Coplien, Booch, Johnson and others was the first step forward to forming a design patterns community in the field of software engineering. In 1995, Erich Gamma,

Richard Helm, Ralph Johnson, and John Vlissides (the Gang-of-Four, GoF) published "*Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma et al., 1995). (Gamma et al., 1995) documented 23 design patterns in their book; one largely used pattern is the Observer.

2. PATTERNS OF HCI: A DEFINITION

The first milestone about patterns in HCI is the workshop organized at CHI conference in 1997. Until 2001, the discussion about patterns in the HCI community where more focused on defining the concept of interaction pattern and its roles. From the most generic to more HCI domain dependant, a HCI 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 interaction technique to a commonly occurring user problem;
- "An invariant solution to address a recurrent design problem within a specific context" (Dix, 1998);
- A general repeatable solution to a commonly-occurring usability problem in interface design or interaction design;
- A solution to a usability problem that 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" (Borchers, 2001).

In essence, patterns of HCI 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 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, 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, including:

- 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. In other words, patterns are an opportunity to bring together a UI design solution and a software implementation solution in the same place.

For example, different implementations are necessary to support variations in design look and feel, platform preference and usage context. For example, 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* (Javahery and Seffah, 2002).

As a conclusion, some important defining characteristics and basic terminologies that are relevant to patterns include: **identification** of the problem in context and with imposed constraints, **existence** of the solution, **recurrence** of the problem, **invariance** abstraction of aspects of the solution, **practicality** of the solution, which needs to strike a balance between **optimality** and **objectivity**, and **communicability** of the problem and the process of arriving at the solution to the user. The relationship between some of these characteristics is illustrated in Figure 1**.**

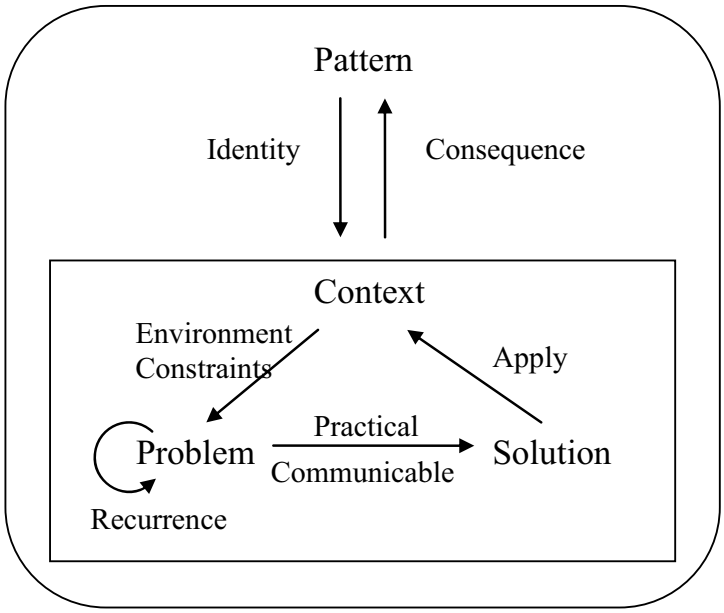

Figure 1. Pattern's Anatomy and Components

3. MISCONCEPTIONS ABOUT DESIHM PATTERNS

Common misconceptions about patterns (Beck et al., 1996) can be summarized as follows:

- Patterns are only object-oriented;
- Patterns provide only one solution;
- Patterns are implementations;
- Every solution is a pattern.

Although most of the **patterns are object-oriented**, patterns can also be found in variety of software systems, independently of the methods used in developing those systems (Beck et al., 1996). Patterns are widely applicable to every software system, since they describe software abstractions (Beck et al., 1996).

Patterns provide more than one solution. Patterns describe solutions to the recurring problems, but do not provide an

exact solution, rather capture more than one solution. This implies that a pattern is not an implementation, although it may provide hints about potential implementation issues. The pattern only describes when, why, and how one could create an implementation.

Every solution is not necessary a pattern. Not every solution, algorithm, or heuristic can be viewed as a pattern. In order to be considered as a pattern, the solution must be verified as recurring solution to a recurring problem. The verification of the recurring phenomenon is usually done by identifying the solution and the problem (the solution solves) in at least three different existing systems. This method of verification is often referred to as the *rule of three*. The following example of (Alexander, 1979) illustrates this misconception:

> **Window place** Consider one simple problem that can appear in the architecture. Let us assume that a person wants be comfortable in a room, implying that the person needs to sit down to really feel comfortable. Additionally, the sunlight is an issue, since the person is most likely to prefer to sit near the light. Thus, the forces of pattern in this example are:
>
> **(i)** The desire to sit down, and
> **(ii)** The desire to be near light. The *solution* to this *problem* could be that in every room the architect should make one window into a *window place*.

Not every pattern can be considered to be a good pattern. There is a set of criteria that a pattern must fulfill in order to be a good one. A pattern encapsulating these criteria is considered to be a good pattern (Gamma et al., 1995; Alexander, 1977; Coplien, 2001):

- A solution (but not obvious);
- A proven concept ;
- Relationships;
- Human component.

Thus, (Gamma et al., 1995; Alexander, 1977; Coplien, 2001) claim, according to the criteria quoted above, that a good pattern should solve a problem, i.e., patterns should capture solutions, not just abstract principles or strategies. A good pattern should be a proven concept, i.e., patterns should capture solutions with a track record, not theories or speculation. A good pattern should not provide an obvious solution, i.e., many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly, which is a necessary approach for the most difficult problems of design. A good pattern also describes a relationship, i.e., it does not just describe modules, but describes deeper system structures and mechanisms. Additionally, a good pattern should contain a significant human component (minimize human intervention). All software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetic and utility.

4. PATTERNS AS A TOOL TO CAPTURE BEST DESIGN PRACTICES

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. There have been many partially successful approaches to collect, represent and deliver best design practices. The most popular ones are:

- Study of exemplars;
- Practice under the instruction of a mentor;
- Design principles to capture the mentor's implicit knowledge;
- Design rationale for organizing application of principles to cases;
- Design guidelines and style guides making principles specific;
- UI toolkits embodying some guidelines.

In the nineties, 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.

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 rationale. They also contain associated scenarios which provide designers with a concrete idea of the context of use. When first introduced, claims were limited in their 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.

5. HCI DESIGN PATTERN LANGUAGES

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 – with the common attributes Context, Problem, Solution, Forces, Related Patterns, and Examples. Secondly, 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, 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. Thirdly, pattern interrelationships should be described. In Experiences language, the relationships between the patterns are mapped and indicated by arrows, creating a sort of “flow” within the language.

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.

6. PATTERN LANGUAGES AND THE USER-CENTRIC DESIGN PROCESS

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 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 Pattern-Supported Approach (PSA) Framework, HCI patterns are used at various levels to solve problems relating to business domains and processes, tasks, structure and navigation, and 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 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 groups generally indicates the order in which they should be used in the design process. 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.

7. PATTERNS-ORIENTED DESIGN

(Javahery and Seffah, 2002) 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 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.

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 sub-categorized into Page manager and Information container patterns. 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 multi-criterion 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 sub-ordinates, but pattern D cannot apply since it is a competitor. The relationships are explained below.

In POD, designers first should follow a POD model. The model acts as a guide for designers in making stepwise design decisions. To illustrate POD modeling, for website design, 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, 2001) and (Welie and Van Der Veer, 2003) 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).

Designers should exploit relationships between patterns. We have described five types of relationships between the UPADE patterns, published in (Taleb et al., 2006; Javahery et al., 2006). The same relationships can easily be applied to other pattern libraries. This multi-criterion 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 X, then Patterns Y and Z apply as sub-ordinates, but pattern S cannot apply since it is a competitor.” The relationships are:.

8. PATTERNS AS REUSABLE BUILDING BLOCKS: STRUCTURAL VERSUS BEHAVIORAL APPROACH

The development of interactive applications using design patterns as reusable design components requires a careful look at composition techniques. Several methods have been proposed for composition. For example, (Yacoub and Ammar, 2003) proposed two composition techniques categorized and illustrated as: Behavioral versus Structural Composition.

Behavioral composition approaches are concerned with objects as elements that play multiple roles, where each role is part of a separate pattern. These approaches are also known in the *OO* literature as interaction-oriented or responsibility-driven composition (Wirfs-Brock and Wilkerson, 1989). Although, the POD composition approach uses notation and composition techniques that are based on the pattern structure (i.e., its class model), (Yacoub and Ammar, 2003) find it useful to be familiar with existing composition techniques that utilize the pattern's behavior model.

Behavioral approaches enable to modeling and composing patterns, while having advantages and drawbacks. Formalizing the behavior specification of individual patterns is important for the purpose of clarifying their semantics and facilitating their utilization by any pattern composition approach. Several authors have proposed various approaches, such as: the approach presented by (Henderson-Sellers et al., 1996) on role modeling and synthesis using the *OO* role analysis method, the works of (Riehle, 1997) presented at the *OOPSLA* conference in 1997. This approach in (Henderson-Sellers et al., 1996; Riehle, 1997) applies the concepts of role models suggested by Henderson-Sellers to pattern composition. Others approaches are presented in the composition field such as the approach called "the superimposition" proposed by (Bosch, 1998), which uses design patterns and frameworks as architectural fragments and merges roles and components to produce applications and finally, another approach three-layer "role/type/class" proposed and developed by (Lauder and Kent, 1998), which takes a visual specification approach to describe design patterns.

Structural composition approaches build a design by gluing pattern structures that are modeled as class diagrams. Structural composition focuses more on the actual realization of the design rather than abstraction, using different types of models, such as role models. Behavioral composition techniques, such as roles (Henderson-Sellers et al., 1996; Riehle, 1997; Kristensen and Østerbye, 1996), leave several choices to the designer with less insight on how to continue to the class design phase. Techniques that consider both structural and behavioral views could be complex and difficult to use. Therefore, the POD approach advocates a structural composition approach with pattern class diagrams (Henderson-Sellers et al., 1996; Riehle, 1997; Kristensen and Østerbye, 1996). Constructional design patterns in which a pattern interface can be clearly specified lend themselves to a structural composition approach (Henderson-Sellers et al., 1996; Riehle, 1997; Kristensen and Østerbye, 1996).

(Yacoub and Ammar, 2003) discussed several structural composition techniques and contrast these techniques with a proposed POD methodology. One approach for pattern-oriented design is proposed by (Ram et al., 1997). In contrast to the top-down approach, this approach describes a bottom-up process to design software using design patterns. This approach shows how related patterns can be selected; however, it does not clearly show how patterns can be composed. Nevertheless, it gives an example of previous attempts in the literature to develop a systematic process for pattern-oriented software development.

9. PATTERN MODELING AND REPRESENTATION

We can look at patterns in general as artifacts that have three main milestones, organized from a user perspective (Figure 2).

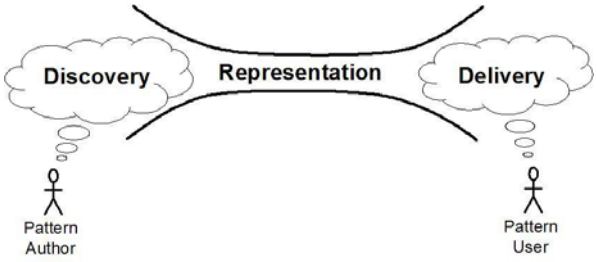

Figure 2. Major milestones and users of patterns

Pattern Delivery

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.

Pattern Discovery

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.

Pattern Representation

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 come 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 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 can not 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. Partial automation of this process, combined with sound experience and good common sense can significantly facilitate the analysis and design phase of software development. 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:

- 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

10. OPEN ISSUES

A universally accepted taxonomy for pattern is still missing in HCI. Patterns deal with different levels of abstraction and have to be considered at different stages. Therefore, if languages are not structured logically, it can be confusing for designers trying to work with them. Some authors have suggested their own partial classifications to facilitate the use of patterns. For example, (Welie, 1999) discusses a taxonomy based on the domain of Web application, GUI or Mobile UI design patterns. (Tidwell, 1997) organizes her patterns according to different facets of UI design; categories include Content Organization, Navigation, Page Layout, and Actions/Commands.

Furthermore, 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.

A further challenge is the lack of tool support, which makes it difficult to capture, disseminate and apply patterns effectively and efficiently. Tools need to be developed with three major objectives in mind. Firstly, tools are needed to support UI designers and software engineers involved in UI development. Secondly, as a research forum for understanding how patterns are really discovered, validated, used and perceived, tools are also required. Thirdly, automation tools are needed to support the usage of patterns as prototyping artifacts and building blocks. The following are some of the required features (Gaffar and Seffah, 2006):

- Tools have to be designed to accept proposed or potential patterns in many different formats or notations. Therefore patterns in versatile formats can be submitted for reviewing;
- A common editorial board for reviewing and validating patterns is also required. Before publishing, collected and contributing, patterns must be accessed and acknowledged by the editorial committee. We are inviting HCI patterns practitioners and researchers to set up and join this committee;
- A pattern ontology editor to capture our understanding of pattern concepts and to put them into relation with each other (Taxonomy) will be an important step toward a systematic usage of patterns as well as the emergence of a pattern-assisted design tool;
- Tools are needed to allow us to attach semantic information to the patterns. Based on this information and our ontology, patterns will be placed in relationships, grouped, categorized and displayed;
- A pattern navigator can also provide different ways to navigate through patterns or to locate a specific pattern. The pattern catalogue can be browsed by pattern groups or searched by keyword. Moreover, a pattern wizard will find particular patterns by questioning the user;
- A pattern viewer will help in providing different views of the pattern, adjusted to the preferences of the specific pattern user's need.

11. REFERENCES

For further information on patterns including an exhaustive list of references, please visit the IPE (Integrated Pattern Environment Website) at hci.concordia.ca