Presented at CHI-SA'2000, 8-10 May 2000, University of Pretoria, South Africa.

The Role of HCI Design Patterns in Software Development in South Africa

Janet Wesson
Department of Computer Science & Information Systems
University of Port Elizabeth
csajlw@upe.ac.za

Abstract

This paper will discuss the role of HCI design patterns in software development in South Africa. The concept of a design pattern, whilst popular in several other fields including architecture and software engineering, is relatively new to Human-Computer Interaction (HCI). The potential of pattern languages as a vehicle for the dissemination of HCI design knowledge is becoming widely recognized within the HCI community. This potential is based on the ideas of the architect Christopher Alexander. Patterns are developed to record the invariant properties that exist in a design solution which attempts to resolve conflicting social, cognitive, and technological forces. Patterns are interlinked into a network (termed a pattern language) to support both conceptual and detailed design.

1. Introduction

The South African user population is characterized by a wide diversity of culture, language, education and technological expertise. This diversity has created a unique set of challenges for the software industry in South Africa. This country has also witnessed an unprecedented growth in the development of information technology in recent years, which has served to aggravate the problem, rather than improve it. Too many South African users find it difficult to learn to use computer systems effectively. The question is why, and what can be done to address this issue?

The most important aspect of modern interactive computer systems is the level of support they provide for the underlying human activity [NL1995]. This level of support is encompassed in the user interface (UI) with which the user interacts with the system. Within the field of HCI, considerable empirical research has been done towards ensuring that the UI is designed according to standard design principles. Despite the use of these guidelines, however, software is still designed with serious usability flaws [Wes1999]. The aim of this paper is to explore the reason behind these design flaws, and propose a solution which will attempt to remedy this situation. This solution is the use of HCI design patterns as an aid for initial UI design.

This paper will discuss the different categories of HCI design patterns and how these can be integrated into the software process in order to design usable systems. An example of a design pattern will also be given in order to illustrate the use thereof.

2. Usability in South Africa

Throughout the software industry, software engineers develop user interfaces (UIs) with little or no support from professional UI designers [Joh2000]. As a consequence of this, commercial software often contains serious design errors. Examples of such software are to

be found both nationally and internationally. Catalogues of poor UI design, or design "bloopers" may be found in several Web sites, including the Interface Hall of Shame (www.iarchitect.com/mshame.htm) as well as Jakob Nielsen's top ten Web design errors (http://www.useit.com/) and a Web site compiled by Vincent Flanders (http://www.webpagesthatsuck.com/).

The situation in South Africa is no better, and is aggravated by a lack of HCI training and expertise in the software industry. A survey by the author in September 1999 revealed that only 7 out of 25 universities (28%) in South Africa are actually teaching HCI principles as part of the typical Computer Science or IT undergraduate or postgraduate program. Consequently, very few software engineers in South Africa have any real knowledge or understanding of the implications of poor UI design. This situation is further complicated by the wide diversity in technological skills possessed by the general user population in South Africa.

An investigation into several interactive systems within South Africa was conducted by the author's postgraduate HCI class earlier this year. This investigation focused on an analysis of the individual likes and dislikes of each system. Students were asked to select a system with which they were familiar and prepare a report on what feature they liked or disliked and why. This investigation revealed that certain serious usability problems exist in software which is in daily use in South Africa.

The systems investigated ranged from telephone and cell-phone based systems, to Ecommerce systems, like Amazon.com and 24.com, as well as ATM systems, photocopier systems, a VCR system and the UPE library system. In all cases, the "dislikes" far outnumbered the "likes". These list of dislikes were then categorized using the 8 golden principles for usability, as discussed in Newman and Lamming [NL1995]. This analysis revealed that the majority of problems lay with the user satisfaction (24%), followed by the frequency of errors and error recovery (24%), the ease of learning and skill retention (21%), and the level of task support provided (18%). Only 6% of usability problems related to speed or issues of customizability respectively.

The problems discovered with the above systems are all inherently usability problems. The International Organization for Standardization (ISO) defines usability as "the effectiveness, efficiency and satisfaction with which specified users can achieve specified goals in particular environments" [Fau2000]. This concept of usability is embodied in the UI with which the user interacts with the system in order to achieve the specified goal.

Traditional academic courses in HCI are not sufficient, however, to ensure good UI design. An investigation by Warren & Viljoen [WV1998] revealed that despite an HCI course and rapid application development tools, novice UI designers still produce poor UI designs. They conclude that experienced UI designers evidently know something that inexperienced designers do not, and that this knowledge has not been adequately articulated in existing HCI literature.

3. Usability Engineering: An Art or Science?

Traditionally, the software development process has tended to fall within the realm of software engineering. The field of software engineering (SE) typically includes the study of different paradigms and methods for software construction as well as software reliability and

maintenance. Early SE methods adopted a data-oriented or process-oriented approach to software design. These approaches were later combined to yield an object-oriented (OO) approach in which data and methods are encapsulated into objects. Whilst research has shown that an OO approach has several advantages such as reusability and extensibility [You1994], [Bah1999], this approach fails to address the UI issue satisfactorily. The author's thesis [Wes1997], highlighted this problem, and showed that truly usable systems can only be developed using a user-centered approach with an early focus on the tasks of the user.

Usability engineering is often interpreted as an approach to the development of software and systems which involves user participation from the outset and guarantees usability by utilizing a usability specification and metrics [Fau2000]. Usability engineering typically includes the design-evaluate-redesign cycle and thus represents the entire process of producing usable software products. Usability engineering and usability evaluation may also be seen as consisting of the components which comprise user-centered design. Usability engineering focuses on the needs of the user and follows a structured, engineering approach in order to ensure that the finished product meets the user's needs.

The basic tenet of usability engineering is "Know the user" and "Know the task" [Fau2000]. In order to do this, designers need to discover who the users of the system will be and study the workplace and work methods. The first stage of requirements specification is to recognize the goal of the system and the problems that exist within the current system. The user requirements must then be translated into a design which solves these problems. This process is, however, not as simple as it is purported to be. Considerable UI design expertise is still required to translate user requirements into a highly usable design [WV1998].

Design guidelines and style guides contain heuristics for designing UIs in different environments. These guidelines typically result from considerable empirical research into what constitutes good UI design (for example see [FS1995]). Whilst a plethora of such guidelines exist, real world usage of these has revealed that many are in fact contradictory and too low level to be of any significant real use to the novice UI designer [VZ1995], [WV1998]. The conclusion is that designers need better advice on how to create an initial design.

4. A Brief History of Patterns

The concept of patterns and pattern languages originated with the work of Christopher Alexander in architecture [Ale1979]. The hypothesis is that one can achieve design excellence by learning and using a carefully designed set of design rules or patterns. The patterns that he and his colleagues defined in [AIS+1977] were an attempt to codify generations of architectural wisdom. They are not abstract principles that require you to rediscover how to apply them successfully, nor are they specific to one particular situation or culture. Instead they are somewhere in between: a pattern describes possible good solutions to a common design problem within a given context, by describing the invariant qualities of those solutions. According to Alexander, an essential component of this approach is that the patterns should be structured and interlinked to form a pattern language.

For example, consider the following Alexandrian pattern: "To solve the problem of creating an *Identifiable Neighbourhood* with *Activity Nodes* and *Public Squares,* the *Street Café* pattern can be applied: Several rooms of a café create a *Gradient of Privacy*, from a terrace with tables near the busy path, to more quiet rooms at the back. Newspapers in one corner make it an inviting place to pass some time, etc."

Note that the pattern is not just proscriptive. It describes something positive, something you can try to build, even though you would naturally vary it according to the particular situation. Note also that it carries values - the value of privacy, the value of reading material. Alexander's goal is not to make a building which is merely trendy, or efficient, or even good-looking; he is looking for ways to create a genuinely good experience for people, via their environment. Alexander's patterns all contain the following: pattern name, forces, examples, solution, an illustration or diagram, and a reference to other patterns.

The patterns concept was introduced to the SE community in 1995 when the "Gang of Four" [GHJ+1995], published a book describing a pattern language for OO software design. A software design pattern is considered to be a proven solution of a recurring SE problem that balances design constraints in a given situation. Like the Alexandrian patterns, these software design patterns provide design solutions that are concrete enough to put immediately into practice, yet abstract enough to apply to many different situations.

The use of patterns in HCI is quite recent. Patterns were first discussed at a CHI'97 workshop on Pattern Languages for Interaction Design [BBC+1997]. Subsequently several authors, including Tom Erickson [Eri1998], Jennifer Tidwell [Tid1998], and Warren and Viljoen [WV1998] have published papers on patterns for UI design. In this context, the use of patterns is seen as being much closer to the aesthetic issues originally found in architectural design than software design patterns. The pattern language approach for UI design thus offers a tool for communication and dissemination of design knowledge within the HCI domain, and with other domains, notably software engineering.

Design guidelines suffer from a number of problems, including being imprecise, having limited applicability, and being contradictory. Pattern languages can form a framework in which HCI knowledge can be structured. Patterns for interaction design thus represent a means of consolidating information in design guidelines. The advantage is that patterns can be easily understood and used by the UI designer.

The potential of pattern languages as a vehicle for UI design was further explored at a workshop held during INTERACT'99 in August 1999, in which the author participated. This workshop served to highlight the goals of a HCI pattern language, as well as identifying the main organizing principles for HCI design patterns. A HCI pattern language was defined as "*A collection of sufficiently complete and ordered patterns to guide the UI designer in designing solutions to problems*".

The key organizing principle identified was that of *scale*, and the categories identified consisted of the following: Society, Multiple Users, Social Position, System, Application, UI Structure or Dialogue, Components, Primitives and Physical Properties.

5. HCI Design Patterns

Each pattern description defines a context of use, a problem the designer needs to solve, a set of "forces" pushing the designer in different directions, and a primary rule, and sometimes additional secondary rules, on how those forces might be resolved to best solve the problem. Examples are also provided, both good and bad. Sometimes the bad examples show inappropriate uses of the pattern, and other times they show a situation in which the pattern should have been used but wasn't. In addition patterns should be flexible and adaptive enough

to generate solutions in varying contexts and reference other patterns to use synergistically in order to create a solution as a whole.

Initially HCI patterns were based on the Alexandrian format used to describe architectural patterns. This format has evolved as the field has matured, and a framework for defining HCI patterns was established at the INTERACT'99 Workshop on Pattern Languages and subsequently refined during the CHI'2000 Workshop in April 2000. This framework is explained below.

A HCI pattern is defined as consisting of the following: a self-explanatory name which describes the solution; a ranking (given as 3 levels of confidence); a sensitizing example (which may be an image, a video or an audio clip); the design context in which it can be applied; a problem statement which is described in terms of the conflicting forces which exist; the design rationale with real examples from different domains; the solution with concise, generalizing examples; an illustration; references to related patterns; a summary; and the name of the author(s).

6. Examples of Pattern Languages

Jennifer Tidwell [Tid1998] has attempted to write a pattern language for HCI design which consists of about 50 design patterns for designing interactive systems. This collection of patterns is widely regarded as currently the most ambitious attempt at a UI pattern language. The patterns in this pattern language address the general problem of how to design a complex interactive software artifact. The goal of the pattern language is to support high-quality interaction between the user and the software artifact.

Examples of patterns covered in this pattern language include patterns on the shape of the content, such as *Narrative* or *High-Information Display*; as well as patterns on the shape of the user actions, such as *Form* or *WYSIWYG Editor*; and patterns on the specific action taken by the user, such as *Choice from a Small Set* or *Forgiving Text Entry.* The *Form* pattern is specified in terms of a number of sub-patterns such as *Choice from A Small Set*, which the designer must use to select the most appropriate control for each piece of information in the form.

The patterns and sub-patterns as proposed by Jennifer Tidwell represent an excellent starting point and framework for patterns-based UI design. These patterns were, however, written prior to the INTERACT'99 workshop at which a format for a HCI pattern was defined. Accordingly the definition of these patterns does not include certain aspects, in particular, graphical examples and illustrations. In addition, the forces and solution for these sub-patterns do not encompass the same ergonomic knowledge as contained within typical design guidelines [VZ1995], [SF1995]. It is our contention, therefore that the *Form* pattern as proposed by Tidwell can be extended to include this knowledge, and that this will represent a simple UI Structure or Dialogue-level pattern language. The modified *Form* pattern, entitled the *Data-Entry Form*, is given in the next section.

7. The *Data-entry Form* Pattern

Name: *Data-Entry Form*
Author: Janet Wesson

Ranking: Good (Reviewed in writers workshop in CHI'2000)
Related Patterns: *Form* (in Tidwell's *Common Ground*)

Example: For an example of the typical use of a data-entry form, see the Microsoft Word 2000 Print Dialogue in Figure 1 below.

Figure 1 Print Dialogue in Microsoft Word 2000

Context: The user has to provide preformatted information, usually short (non-narrative) answers to questions.

Problem: How should the artifact indicate what kind of information should be supplied, and the extent of it?

Forces:

- It should be clear what the user is supposed to supply (input) and what is output (display) only.
- The user needs to know what kind of information to provide.
- The user needs to know what is required, and what is optional.
- The range of possible values entered may be known, unknown, or a combination of known and unknown (domain of input).
- The number of possible values entered may be limited, or unlimited (Npo).
- The user may enter one or more values (Nvc).
- The display can consist of a single item or multiple items (Nvd).
- The display is limited to some physical length (Lg).

Solution: Provide appropriate "blanks" to be filled in, which clearly and correctly indicate what information should be provided. Visually indicate those editable blanks consistently, such as with subtle changes in background color, so that a user can see at a glance what needs to be filled in. Label them with clear, short labels that use terminology familiar to the user; place the labels as close to the blanks as is reasonable. Arrange them all in an order that makes sense semantically, rather than simply grouping things by visual appearance.

Resulting Context: You must select controls or *interaction objects* (IOs) for each piece of information to be supplied. These may include:

- *Single Choice*, if the user can select or not select a single item.
- *Choice*, if the user must pick one from a set of two to five choices.
- *List Selection*, if the user should pick one from a set of 10 or fewer choices

- *Long List Selection*, if there are more than 10 choices.
- *Single Text Entry*, if the text will occupy not more than one line and the domain of the text to be supplied by the user is unknown.
- *Multiple Text Entry*, if the text will occupy more than one line and the domain of the text to be supplied by the user is unknown.

Illustration: Figure 2 contains a snapshot of a job application form that illustrates the use of the six sub-patterns given in the *Data-Entry Form* pattern as described above.

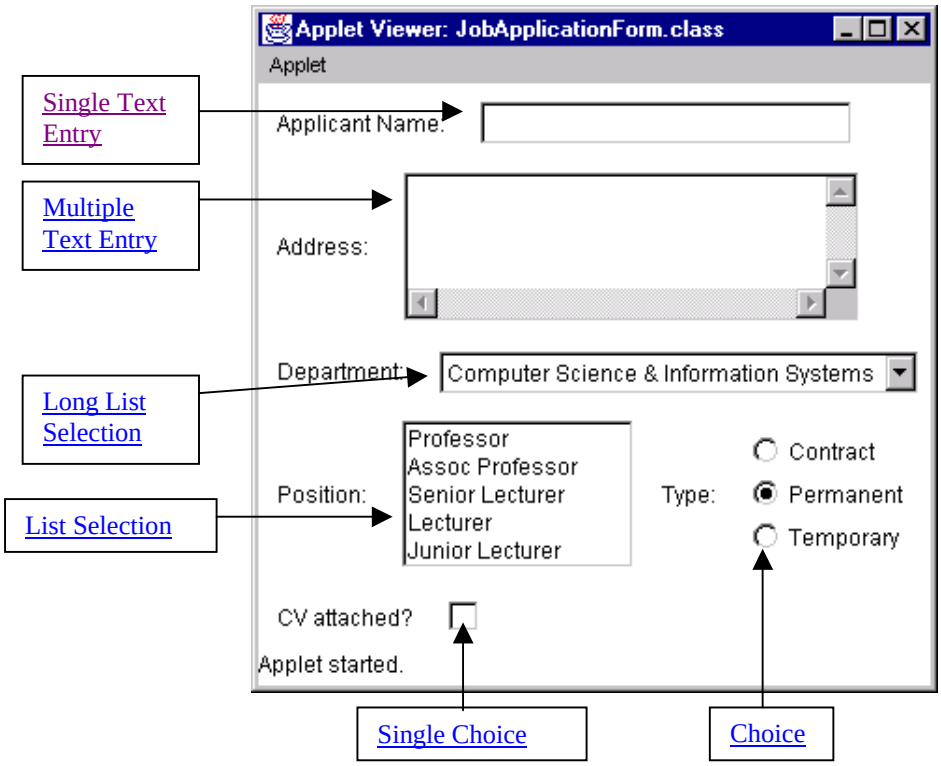

Figure 2: An Example of the Data-Entry Form Pattern: A Job Application Form

7.1 The Sub-Patterns

Name: *Single Choice*
Problem: How can a choice or non-choice of a single item on a screen be best displayed?
Context: The user wants to indicate a choice or non-choice of a single item on a screen.
Forces:
- The domain of input.
- The number of possible values that can be entered (Npo).
- The number of values to be chosen (Nvc).
- The screen density.

Examples:
- Font Effects selection in Microsoft Word.
- A boolean value in an application form, for example is a CV attached? (Figure 2).

Solution: Use a *Single Choice* pattern to present a choice or non-choice of a single item if the domain is known, the number of possible values to be entered (Npo) is 1, the number of possible values to choose (Nvc) is 1, and the screen density is high.

Design rationale: If the domain is known and the Npo is 1 then a *Single Choice* pattern will minimize keystrokes and reduce the number of possible user input errors. If the screen density is low, then a *Choice* pattern may be used instead (i.e. Yes/No).

Name: *Choice*
Problem: How can a choice of one of two or more items on a screen be best displayed?
Context: The user wants to indicate a choice of one of two or more items on a screen.
Forces:

- The domain of input.
- The number of possible values (Npo).
- The number of values to be chosen (Nvc).
- The screen density.

Examples:

- The selection of the type of document in the New Document dialogue in Microsoft Word.
- The selection of the type of employment in an application form as contract, permanent or temporary (Figure 2).

Solution: Use a *Choice* pattern to present a choice of two to five items if the domain is known, the number of possible values to be entered (Npo) is greater than 1 and less than 6, the number of possible values to choose (Nvc) is 1, and the screen density is low.

Design rationale: If the domain is known and the Npo is greater than 1 and less than 6, then a *Choice* pattern will minimize keystrokes and reduce the number of possible user input errors. If the screen density is high, then either a *List Selection* pattern or a *Long List Selection* pattern may be used instead.

Name: *List Selection*
Problem: How can a short (up to about 10 items) list of choices be best displayed?
Context: The user wants to select an item from a list of items (which is less than some ceiling) or wants to view a list of items, without the capability of item choice.
Forces:

- The domain of input.
- The number of possible values (Npo).
- The number of values to be chosen (Nvc).
- The screen density.

Examples:

- Font type selection in Microsoft Word.
- Selecting the position applied for in a job application form (Figure 2).

Solution: Use a *List Selection* pattern to present a choice of 10 or fewer choices if the domain is known, the number of possible values to be entered (Npo) is greater than 5 and less than 11, the number of possible values to choose (Nvc) is 1 and the screen density is low.

Design rationale: If the domain is known and the Npo is greater than 5 and less than 11, then a *List Selection* pattern will minimize keystrokes and reduce the number of possible user input errors. If the screen density is high, then a *Long List Selection* pattern may be used instead.

Name: *Long List Selection*
Problem: How can a long (more than 10 items) list of choices be best displayed?
Context: The user wants to select an item from a long list of items (which is greater than some ceiling) or wants to view a long list of items, without the capability of item choice.
Forces:
- The domain of input.
- The number of possible values (Npo).
- The number of values to be chosen (Nvc).
- The screen density.

Examples:
- The Alignment drop-down list in the Format Paragraph dialogue in Microsoft Word.
- The selection of a department from a drop-down list of departments in a job application form (Figure 2).

Solution: Use a *Long List Selection* pattern to present a choice of more than 10 items if the domain is known, the number of possible values that can be entered (Npo) is greater than 10, the number of possible values to be chosen (Nvc) is 1 and the screen density is high.

Design rationale: If the domain is known and the Npo is greater than 10, then a *Long List Selection* pattern will minimize keystrokes and reduce the number of possible user input errors. If the screen density is low, then a *List Selection* pattern may be used instead.

Name: *Single Text Entry*
Problem: How can a short (single line) text object be best displayed?
Context: The user wants to enter a single text object that spans not more than one line in a window, or wants to view the same, with the capability of editing it if desired.
Forces:
- The domain of input.
- The number of possible values (Npo).
- The number of values to be chosen (Nvc).
- The physical length of the data to be input or displayed.

Examples:
- Entering the Subject of an e-mail message in Microsoft Outlook.
- Entering a range of page numbers to print in the Print dialogue in Microsoft Word.
- Entering an applicant name in a job application form (Figure 2).

Solution: Use a *Single Text Entry* pattern to enter or display a single item if the domain is unknown and the length of the text is less than or equal to Lg.

Design rationale: If the domain is unknown then a *Single Text Entry* pattern is the only alternative to enter or display a single item. If the length of the text is greater than Lg, then a *Multiple Text Entry* pattern may be used.

Name: *Multiple Text Entry*
Problem: How can a long (multi-line) text object be best entered or displayed?
Context: The user wants to enter a single text object that spans more than one line in a window, or wants to view the same, with the capability of editing it if desired.

Forces:
- The domain of input.
- The number of possible values (Npo).
- The number of values to be chosen (Nvc).
- The physical length of the data to be input or displayed.

Examples:
- The Font Preview text area in the Format dialogue in Microsoft Word.
- The Message Body text area in the new message dialogue in Microsoft Outlook.
- The Address text area in a job application form (Figure 2).

Solution: Use a *Multiple Text Entry* pattern to enter or display multiple items if the domain is unknown and the length of the text is greater than Lg.

Design rationale: If the domain is unknown and the length of the text is greater than Lg, then a *Multiple Text Entry* pattern is the only alternative to enter or display a single item.

8. The Role of Patterns

This section will discuss the role of patterns in UI design. Certain key questions need to be answered. These include, inter alia, who are the real users of patterns? What are patterns used for? When can patterns be used? How does the designer know which patterns to use? And how can the designer know that certain patterns can be trusted to produce a good UI design?

Patterns are intended for a number of different users and at a number of different knowledge levels. Similarly to architecture, the visual aspect of a HCI pattern serves as a communication tool between the end user and the designer. The end user can then easily visualize via the accompanying diagram exactly what the final product would look like before it is constructed. The UI designer, on the other hand, can use the pattern as an aid for initial design at a conceptual level which is significantly higher than design guidelines. Patterns can also be used in HCI education as a means of quickly and easily conveying UI design knowledge to students and software developers. At a higher level, pattern writers can use patterns to synthesize the design knowledge inherent in successful designs.

The role of patterns is closely linked to the requirements of the individual users. For end users, patterns serves as a communication tool. For software developers, patterns serve as a design tool. For students and educators, patterns can be used as an educational tool. And finally, for pattern writers, patterns can represent a resource tool for collecting and synthesizing good design practice.

HCI Patterns can be used almost throughout the software development process in order to strongly support user-centered design. Several authors have referred to this as a patterns-based approach to software development. At the requirements analysis or information gathering stage, task-based patterns can be used to illustrate the software developers understanding of the users' requirements. HCI design patterns can then be used to effectively bridge the gap between requirements and design by providing an initial view of what designs have worked successfully in the past within a given context to solve a specified problem. In many cases, the use of these patterns may actually shorten the time required or circumvent the need to prototype the initial design. As there may be several alternative patterns within a given context, the user will be able to easily visualize the available alternatives and choose the

one which meets his needs better. This may have a significant impact on the user satisfaction, and inherently the usability of the final software.

Design patterns can also be used during the evaluation stage and as an aid for system maintenance since they represent the design rationale used to design the UI. In addition, design patterns may be used as a repository of design knowledge since they encompass examples of good design practice.

The question for the UI designer, however, is how does he locate the relevant patterns and having located the patterns, how does he know which to use in a given situation? At this early stage of pattern development, a good place to start is to look at existing pattern collections, such as those provided by Tidwell [Tid1998], Erickson [Eri1998], and others. Links are provided to these and other pattern collections from the CHI'2000 Workshop web page (see www.it.bton.ac.uk/staff/rng/CHI2K_PLworkshop/). An international community of pattern writers has been established, and several publications on design patterns are envisaged shortly. The advantage of patterns, however, is that the user can become involved in the actual selection process.

Finally, there is the issue of confidence and reliability. How can the UI designer rely and trust on the design knowledge embodied in a specific pattern? This question has been addressed by including the authoring history, evidence of usage, a confidence factor, and theoretical references within each pattern.

9. The Way Forward?

HCI patterns focus on patterns that improve the *quality in use* of the system. Just like Alexander's patterns, the goal is to focus on patterns that improve the "quality without a name" for the people that use the software artifacts - the *users*. Therefore, all patterns must show their benefit for users by arguing what their impact on *usability* is.

This paper has discussed the role of patterns in UI design and presented some example patterns. In order to establish a *Pattern Language*, the patterns should be seen as 'candidates'. These patterns should be discussed and developed further by a community of pattern writers and software developers.

Certain research questions still exist for HCI patterns. These include how best to find (*harvest*) HCI design patterns; how to document them; how to maintain pattern languages; how to disseminate pattern languages; how to apply patterns; how to evaluate patterns; how to evaluate the application of patterns; and how to teach pattern use. We also need a framework for classifying and writing patterns. More case studies and examples of patterns are also needed, particularly for different domains.

10. Conclusions

Whilst considerable research has already been done in the field of pattern languages, much must still be done [WV1998]. More research is necessary to develop new pattern languages and to refine existing patterns [WC2000]. These pattern languages will also need to be validated for different application domains within the South African software context. In principle, however, design patterns may represent a solution to the problem of designing usable computer systems in South Africa.

References

Alexander, Christopher, Ishikawa, Sara, Silverstein, Murray, and Jacobson, Max (1977). *A Pattern Language*, New York: Oxford University Press, 1977

Alexander, Christopher (1979). *A Timeless Way of Building*. New York: Oxford University Press. 1979.

Bahrami, Ali. (1999). *Object Oriented Systems Development*. McGraw-Hill.

Bayle, Elizabeth, Bellamy, Rachel, Casady, George, Erickson, Tom, Fincher, Sally, and Thomas, John (1997) *Putting it all together: Towards a Pattern Language for Interaction Design. Summary of CHI Workshop*. http://www.pliant.org/personal/Tom_Erickson/Patterns.WrkShpRep.html.

Erickson, Tom. (1998) *The Interactions Design Patterns Page*. http://www.pliant.org/personal/Tom_Erickson/InteractionPatterns.html.

Faulkner, Xristine. (2000*).Usability Engineering*, Macmillan Press, 2000.

Fowler, Susan, and Stanwick, Victor R (1995). *The GUI Style Guide*. Cambridge, AP Professional, 1995.

Gamma, Erich, Helm, Richard, Johnson, Ralph, and Vlissides, John. (1995) *Design Patterns: Elements of Reusable Object Oriented Software*. Reading, MA: Addison-Wesley, 1995.

Johnson, Jeff. (2000*).GUI Bloopers*, Morgan Kaufmann, 2000.

Newman, W.M. and Lamming, M.G. (1995) *Interactive Systems Design*, Addison-Wesley, 1995.

Tidwell, Jennifer (1998). *Common Ground: A Pattern Language for Human-Computer Interface Design.* http://www.mit.edu/~jtidwell/common_ground.html (last updated June 1999).

Vanderdonckt, J., Zucchinetti, G. (1995). *Key Activities for a Development Methodology for Interactive Applications*. In D.Benyon and Ph. Palanque (editors), Critical Issues in User Interface Systems Engineering, chapter 7. Springer-Verlag.

Warren, P. and Viljoen, M. (1998) *Design patterns for user interfaces*. In Proceedings of SAICSIT'98, November 1998 (http://saturn.cs.unp.ac.za/~peterw/papers/patterns.htm).

Wesson, Janet L. (1997). *An Investigation into Design Methodologies for Usability: A Case Study Approach*. PhD thesis, University of Port Elizabeth (http://www.cs.upe.ac.za/staff/csajlw/Publications/PhdAbstract.HTM).

Wesson, J. (1999) *Integrating HCI with the Software Development Process.* In Proceedings of BitWorld'99, Cape Town, 30 June-2 July 1999 (http://www.cs.upe.ac.za/csajlw/Publications/IntegratingHCI.htm).

Wesson, J. and Cowley, N.L. (2000) *Selecting Interaction Objects: A Patterns Approach.* Presented at the CHI'2000 Workshop on Pattern Languages for Interaction Design, The Hague, 2-3 April 2000.

Yourdon, E. (1994). *Object-Oriented Systems Development: An Integrated Approach*. Prentice-Hall.