HUMAN–COMPUTER INTERACTION, 2006, Volume 21, pp. 49–102

Pattern Languages in HCI: A Critical Review

Andy Dearden
Sheffield Hallam University

Janet Finlay
Leeds Metropolitan University

ABSTRACT

This article presents a critical review of patterns and pattern languages in human–computer interaction (HCI). In recent years, patterns and pattern languages have received considerable attention in HCI for their potential as a means for developing and communicating information and knowledge to support good design. This review examines the background to patterns and pattern languages in HCI, and seeks to locate pattern languages in relation to other approaches to interaction design. The review explores four key issues: What is a pattern? What is a pattern language? How are patterns and pattern languages used? and How are values reflected in the pattern-based approach to design? Following on from the review, a future research agenda is proposed for patterns and pattern languages in HCI.

Andy Dearden is an interaction designer with an interest in knowledge sharing and communication in software development. He is a senior lecturer in the Communication and Computing Research Centre at Sheffield Hallam University. **Janet Finlay** is a usability researcher with an interest in design communication and systems evaluation. She is Professor of Interactive Systems in Innovation North at Leeds Metropolitan University.

CONTENTS

1. INTRODUCTION
2. THE SCOPE OF THIS REVIEW
    2.1. General Software Design Patterns
    2.2. Interface Software Design Patterns
    2.3. Interaction Design Patterns
3. A SHORT HISTORY OF PATTERNS
    3.1. Christopher Alexander
    3.2. Pattern Languages in Software Engineering
    3.3. Patterns in HCI
4. ISSUE 1: WHAT IS A PATTERN?
    4.1. Characteristics of Pattern
    4.2. Identifying Patterns
    4.3. The Presentation of Patterns
    4.4. Patterns, Guidelines, and Claims
5. ISSUE 2: WHAT IS A PATTERN LANGUAGE?
    5.1. Pattern Languages and Pattern Catalogues
    5.2. The Organisation of Pattern Languages
    5.3. Notions of Generativity
6. ISSUE 3: HOW ARE PATTERNS AND PATTERN LANGUAGES USED?
    6.1. Patterns for Participatory Design
    6.2. Patterns as Technical Lexicon
    6.3. Patterns as Organisational Memory
    6.4. Patterns as Lingua Franca
    6.5. Patterns as Design Rationale
7. ISSUE 4: VALUES AND PATTERN LANGUAGES
    7.1. The Properties Examined to Identify Patterns
    7.2. Values in the Selection of and Rationale for Individual Patterns
    7.3. Values in the Process of Developing Patterns
    7.4. Values in the Process of Using Patterns
8. CONCLUSION: A RESEARCH AGENDA FOR PATTERNS IN HCI
    8.1. Exploring Pattern Languages in Use
    8.2. Organising Pattern Languages
    8.3. Improving the Production of Pattern Languages
    8.4. Examining Our Values

1. INTRODUCTION

A pattern may be defined as a structured description of an invariant solution to a recurrent problem within a context. A pattern language is a collection of such patterns organized in a meaningful way. In recent years patterns and pattern languages have attracted increasing attention in human–computer interaction (HCI) for their potential in recording and communicating

design knowledge and supporting the design process. Patterns and pattern languages are now being developed and presented in a wide range of HCI areas, including ubiquitous systems (Landay & Borriello, 2003; Roth, 2002), Web design (Graham, 2003; van Duyne, Landay, & Hong, 2003), safety-critical interactive systems (Hussey, 1999), multimedia exhibits (Borchers, 2001a), hypertext and hypermedia (Nanard, Nanard, & Kahn, 1998; Rossi, Lyardet, & Schwabe, 1999; Rossi, Schwaber, & Garrido, 1997; Rossi, Schwabe, & Lyardet, 2000), personal digital assistants (Wier & Noble, 2003), sociotechnical systems (Thomas, 2003), and games design (Bjork & Holopainen, 2004), as well as more general interaction design languages (Laakso, 2003; Tidwell, 1998, 1999a, 2003; van Welie, 2002–2005).

Initial efforts exploring patterns tended to focus on specific pattern development, leading to repeated debates on correctness and commonality of form and structure, together with a certain amount of "partisanship" regarding particular pattern approaches. Work in software engineering and in interaction design shows a variety of debates about the nature of "patterns." Various common elements are generally agreed to be relevant parts of the presentation of patterns, but different authors give significantly different emphases. The result of this is a field that can be daunting to the newcomer, who may find it difficult to disentangle the conceptual characteristics of the approach and therefore its potential contribution to HCI.

In this article we present a critical review of research on patterns and pattern languages in HCI, highlighting four key issues within the field. Our aim is to provide an overview of the field, and identify key literature that may be useful and informative to HCI practitioners and researchers. This review also aims to locate patterns in relation to other established and emerging techniques in interactive systems design, such as guidelines and heuristics (Nielsen, 1994; Smith & Mosier, 1986), style-guides (e.g., GNOME Project, 2003; Microsoft Corporation, 2003); participatory design (Greenbaum & Kyng, 1991; Muller, Haslwanter, & Dayton, 1997; Schuler & Namioka, 1993), claims analysis (Sutcliffe 2000; Sutcliffe & Carroll, 1999) and design rationale (MacLean, Young, Bellotti, & Moran, 1991).

We begin by outlining the scope of the pattern endeavour that we will consider. We then present a short history of patterns, beginning with Alexander's exposition in architecture, through work in software engineering, to the consideration of patterns in HCI, to place the latter in its historical context. Our review then examines different interpretations of the concept of pattern, different ideas on the nature of pattern language, different approaches to the use of patterns within the design process, and different ideas about the role of values in pattern-supported design, before suggesting an agenda for future research.

2. THE SCOPE OF THIS REVIEW

This review is addressed to practitioners and researchers in HCI. Consequently, the primary focus is on patterns and pattern languages that discuss interaction and interface design issues. There are, however, a large number of patterns from other domains (e.g., software engineering and organisational design) that may have a bearing on interactions between humans and computers. To avoid extending the scope of our review beyond practical limits, we define three broad classes of software-related pattern and pattern language that are discussed in the following.

2.1. General Software Design Patterns

A problem is stated in terms of desirable qualities of the internal structure and behaviour of software, and the solution is stated in terms of suggested code structures. The majority of patterns in Gamma, Helm, Johnson, and Vlissides (1995) fall into this category.

2.2. Interface Software Design Patterns

A problem is stated in the domain of desirable interaction behaviours, and the solution is stated in terms of suggested code structures. Examples in this category include patterns for implementing systems that follow a "tools and materials" metaphor (Riehle & Zullighoven, 1995), patterns for implementing digital sound synthesis systems (Judkins & Gill, 2000), patterns to implement queuing of interaction events (Wake, Wake, & Fox, 1996), patterns for e-commerce agent systems (Weiss, 2001), and patterns for mobile services (Roth, 2002).

2.3. Interaction Design Patterns

A problem is stated in the domain of human interaction issues, and the solution is stated in terms of suggested perceivable interaction behaviour. A good example in this category is Tidwell's (1998, 1999a) pattern collection, including patterns such as Go Back to a Safe Place, which advocates providing users with a clearly identifiable way of returning a system to a well-known state, such as the home page of a Web site.

Borchers (2001a) includes three distinct pattern languages, the second of which is composed of interaction design patterns and the third of interface software design patterns. Two examples from Borchers serve to clarify the distinction between interaction design patterns and interface software design patterns. The interaction design pattern Easy Handover deals with the problem that:

> Most interactive systems implicitly assume that each user begins using their system from a start page or initial state. At interactive exhibits, however, one user often takes over from the previous one, possibly in the middle of the interaction, and without knowing the interaction history of the previous user. (Borchers, 2001, p. 117)

The pattern is illustrated by a photograph of two visitors to an interactive exhibit, one who is using the exhibit and another who is waiting for her turn.

The pattern then discusses the design issues and makes the following recommendation:

> Therefore: Minimize the dialogue history that a new user needs to know to begin using an interactive exhibit. Offer a simple means to return the system to its initial state. If critical, user-specific parameters such as language need to be set by a user, let the users change the setting at any time, no matter where they are in the system. (Borchers, 2001, p. 119)

This solution is then illustrated using a "stick figure" drawing.

In contrast, Branching Transformer Chain, an example of an interface software design pattern, takes as its problem:

> If a software system is to react interactively to incoming musical data, it has to perform various processing steps on this data. However, the way in which these processing steps are to be combined is not always obvious. (Borchers, 2001, p. 153)

Here the proposed solution is:

> Therefore: Use a chain of software objects that process the incoming musical data in sequence. Order the transformations so that coarse rhythmic, harmonic or melodic changes are applied before finer-grained adjustments. (Borchers, 2001, p. 155)

The solution is then illustrated by means of a block diagram.

Based on these definitions, this review is primarily concerned with "Interaction Design Patterns" and to a lesser extent with "Interface Software Design Patterns." To set the review in context, it is necessary to consider other literature, particularly from Software Engineering and Architecture. However, within such literature, this review will be restricted to general discussions of pattern languages, rather than discussions of the detailed content of the patterns themselves. Due to space constraints, and the authors' desire to consider the wide range of different approaches to patterns, the article does not include detailed illustrations of any complete patterns. The reader is referred to published pattern languages and collections (e.g., Alexander et al., 1977; Gamma

et al., 1995; van Duyne et al., 2003) or the available online collections of interaction design patterns (e.g., Brighton Usability Group, 2003; Tidwell, 1998, 1999a, 2003; van Welie, 2002–2005). Fincher's Pattern Gallery[1] (Fincher, 2000b) provides an extensive online catalogue of pattern formats from a range of sources, together with example patterns to illustrate each style. To help orientate the reader who is unfamiliar with patterns, we discuss an example of an Alexandrian pattern in the next section.

The early work of Alexander and colleagues (Alexander 1979, 1982; Alexander, Davis, Martinez, & Corner, 1985; Alexander et al., 1977; Alexander, Neis, Anninou, & King, 1987; Alexander, Silverstein, Angel, Ishikawa, & Abrams, 1975) in developing pattern languages in architecture will be considered in order to locate HCI patterns within an appropriate historical context.

3. A SHORT HISTORY OF PATTERNS

3.1. Christopher Alexander

Design patterns and pattern languages arose in architecture from the work of Christopher Alexander and his colleagues. Within his profession his proposals have been controversial (Dovey, 1990; Saunders, 2002) but nonetheless they have captured the public imagination with regard to architecture (Gabriel, 1996b; King, 1993; Saunders 2002) and have been influential in several other domains.

Alexander's early work, summarised in "Notes on the Synthesis of Form" (Alexander, 1964), proposed a systemic approach to architectural design problems. The approach involves analytic decomposition of the problem into subproblems, each characterized by a set of competing forces. By resolving the forces in each subproblem, and synthesizing the individual solutions, the architect generates a solution to the original global problem. Alexander (1964) even considered the possibility of a computational solution to such problems.

During the period from the mid-1960s to mid-1970s, Alexander became skeptical of his suggestions in "Notes on the Synthesis of Form." In the 1970s and early 1980s, he and his colleagues set out to define a new understanding and a new approach to architectural design. Grabow (1983), in his biography, describes the changes in Alexander's thinking during this period as a "paradigm shift." The new approach, centred on the concept of pattern

1. The gallery is available at http://www.cs.kent.ac.uk/people/staff/saf/patterns/gallery.html.

languages, is described in a series of books, namely *The Timeless Way of Building* (Alexander, 1979), *A Pattern Language* (Alexander et al., 1977), *The Oregon Experiment* (Alexander et al., 1975), *The Linz Café/Das Kafe Linz* (Alexander, 1982), *The Production of Houses* (Alexander et al., 1985), and *A New Theory of Urban Design* (Alexander et al., 1987). The books were published as a series, and are explicitly given volume numbers, which do not correspond with the chronological order of publication. Volume 1 of the series (*The Timeless Way of Building*) sets out Alexander's view of how patterns and pattern languages evolve, and how they should be utilized in design. Volume 2 (*A Pattern Language*) offers one instance of a pattern language. The last four volumes of the series each recount a case study in which the pattern based approach to design was applied.

Alexander's pattern language used a specific format for the presentation of a pattern. To illustrate this format, and give a further illustration of the pattern concept, we outline one of his patterns. An Alexandrian pattern starts with the name and reference number (e.g., Light on Two Sides of Every Room, pattern 159). The name is concise and evocative but not obscure. This is followed by a picture (in Alexander's case a photograph) showing an example of an instantiation of the pattern and a short paragraph, which sets its context, including the names of patterns to which this one contributes. The problem that this pattern addresses is then stated. In the case of Light on Two Sides of Every Room, the problem is that:

> When they have a choice, people will always gravitate to those rooms which have light on two sides, and leave the rooms which are lit only from one side unused and empty. (Alexander et al., 1977, pattern 159)

This concise statement of the problem is followed by a detailed discussion and rationale, including the empirical background and evidence (the motivation for the pattern) and the "forces" involved in the resolution of the problem. The solution is then included:

> Locate each room so that it has outdoor space outside it on at least two sides, and then place windows in these outdoor walls so that natural light falls into every room from more than one direction. (Alexander et al., 1977, pattern 159)

The pattern then includes a diagram, which illustrates the solution, and a paragraph indicating how this pattern relates to other "lower" patterns in the pattern language, that is those that contribute to it.

The Alexandrian form has been adopted by some, but by no means all, pattern writers in other fields. For a summary of alternate forms, see the Pattern Gallery (Fincher, 2000b).

3.2. Pattern Languages in Software Engineering

In the late 1980s and early 1990s, researchers in software engineering were exploring ways to reuse design knowledge. For example, Coplien (1992) investigated idiomatic styles of C++ code; Wirfs-Brock, Vlissides, Cunningham, Johnson, and Bollette (1991) examined the design of frameworks that supported effective code reuse; Garlan and colleagues investigated the reuse of formal specifications for a family of products (Garlan & Delisle 1990) and generic software architectures that could be refined to specific implementations (Galan & Notkin 1991; Garlan & Shaw 1993). Alexander's concept of "design patterns" was noticed in the context of this research (Anderson, 1993; Anderson, Coad, & Mayfield, 1994; Beck & Cunningham, 1987; Coad, 1992; Coad & Mayfield, 1993; Gamma, Helm, Johnson, & Vlissides, 1993). The first conference on "Pattern Languages of Programming" (PLoP) was held in August 1994 (Coplien & Schmidt, 1995). Since then, PLoP conferences have been held annually (Harrison, Foote, & Rohnert, 1999; Martin, Reihle, & Buschmann, 1997; PLoP, 1998, 1999, 2000, 2001, 2002, 2003; Vlissides, Coplien, & Kerth, 1996). Other conference series investigating pattern languages in software engineering have also been established (e.g., EuroPLoP in Europe, ChiliPLoP in Arizona, and KoalaPLoP in Australasia). Another important milestone was the publication of Gamma et al. (1995), often referred to as the "Gang of Four" book, which remains one of best selling books in software engineering.

3.3. Patterns in HCI

Early work on patterns in software engineering included solutions for user-interface software design. Thus, Gamma et al. (1993, 1995) included patterns such as Observer (an abstraction similar to the "Model View Controller" architecture) and Decorator (a software design solution used for embellishments such as scrollable panels). The proceedings of the first meeting of PLoP begin with two papers presenting a single interaction design pattern (Adams, 1995) and a pattern language with four interaction design patterns to describe a "tools and materials" metaphor for user interface design, and seven interface software patterns that help implement such interfaces (Riehle & Zullighoven, 1995).

In the proceedings of the third meeting (Martin et al., 1997), user-interface patterns were recognized as a discrete area of interest and afforded a separate "part" of the proceedings, despite being represented by a single paper (Bradac & Fletcher, 1997). In the fourth meeting, four papers were grouped in the proceedings as relating to "Patterns of Human–Computer Interaction" (see Harrison et al., 1999). In 1998 (PLoP, 1998) the organisers grouped the papers using section titles taken from *A Pattern Language*, with the majority of interaction design patterns appearing in the session "Zen View" (pattern 134

in Alexander et al., 1977). Eight of the papers at the 1998 conference include interaction design or interface software design patterns. In 1999 (PLoP, 1999), four papers addressing user-interface issues appear in a group together with two patterns that are primarily concerned with network performance issues. In recent years PLoP has included only a small number of examples of interaction design patterns.

While the number of interaction design and interface software design patterns appearing in PLoP was falling, interest in patterns at meetings of the HCI community was growing. Patterns workshops have become regular events at CHI (Bayle et al., 1998; Fincher et al., 2003; Griffiths, Pemberton, Borchers, & Stork 2000; Schümmer, Borchers, Thomas, & Zdun, 2004; van Welie, Mullet, & McInerney 2002), as well as being held at a meeting of the Usability Professionals Association in 1999 (Granlund & Lafreniere, 1999a), and at Interact in 1999 (Griffiths, Pemberton, & Borchers, 1999). Panels were held at CHI 2001 (Borchers & Thomas, 2001) and at IHM-HCI 2001 (Griffiths & Pemberton, 2001). An early mention of patterns in the mainstream HCI literature was in Norman and Draper (1986) and Norman (1988) but in neither case was the potential use of patterns explored in any detail. More recently, however, papers discussing the use of patterns have been published in a variety of forums including DIS (Erickson, 2000a), ECSCW (Martin, Rodden, Rouncefield, Sommerville, & Viller, 2001), CHI (Dearden, Finlay, Allgar, & McManus, 2002a), PDC (Dearden, Finlay, Allgar, & McManus, 2002b; Schuler, 2002), HCI (Finlay, Allgar, Dearden, & McManus, 2002) and ACM Hypertext conferences (Nanard et al., 1998; Rossi et al., 1997).

A number of interaction design pattern languages have also been published in book form, including Borchers' triple languages for the development of interactive exhibits (Borchers, 2001a), van Duyne et al.'s Design of Sites language (van Duyne et al., 2003) and, most recently, Graham's (Graham, 2003) language on Web usability, as well as many more Web-based collections (e.g., Laakso, 2003; Tidwell, 1998, 1999a, 2003; Van Welie, 2002–2005). These developments are consistent with the expectations of the participants in the early PLoP meetings. In their introduction to the proceedings of the first PLoP conference, Johnson and Cunningham (1995) stated their expectation that "as the PLoP community grows and matures … PLoP will itself splinter along traditional lines of interest" (p. ix).

The remainder of this article will consider four of the key issues that arise within patterns research. We begin with the fundamental question of what is a pattern.

4. ISSUE 1: WHAT IS A PATTERN?

The debate as to what constitutes a pattern has occupied considerable attention in software engineering and HCI. Lea (1994) described the term *pattern* as a

"preformal construct," noting that Alexander provides no formal definition. Alexander offers many different descriptions of patterns that are taken up by different authors. Coad (1992) emphasised the idea of patterns emerging from repetitions in human behaviour, quoting Alexander's observation that "every place is given its character by certain patterns of events that keep on happening there" (Alexander, 1979, as quoted by Coad, 1992, p. 152). Gabriel (1996b), Denning and Dargan (1996), Cline (1996), Johnson and Cunningham (1995), and Borchers (2001a) also highlighted this view. This viewpoint emphasises patterns as recurrent phenomena or structures that must be observed and discovered. The Pointer project (Martin et al., 2001; Martin, Rouncefield, & Sommerville, 2002) captured just such recurrent phenomena, drawing on examples of common interactions derived from ethnographic studies.

An alternative view highlights patterns as artefacts for the explicit representation of design guidance. Gamma et al. (1995) quoted Alexander, "Each pattern describes a problem … and then describes the core of the solution" (p. 2). Beck et al. (1996) described patterns as "a particular prose form" (p. 103) and Borchers (2001b) described patterns as "above all, a didactic medium for human readers" (p. 361). Schmidt, Fayad, and Johnson (1996) and Astrachan, Berry, Cox, and Mitchener (1998) have a similar emphasis.

For Alexander, there is no contradiction between these views. In *The Timeless Way of Building*, Alexander (1979) posited pattern languages as fundamental to the organisation of building, concluding that, "nothing is made without a pattern language in the maker's mind; and what that thing becomes, its depth, or its banality, comes also from the pattern language in the builder's mind" (p. 224). Later, he argued, "in a period when languages are no longer widely shared … it becomes necessary to make patterns explicit … so that they can be shared in a new way—explicitly instead of implicitly—and discussed in public" (Alexander, 1979, p. 246). His effort to explicate patterns gives rise to *A Pattern Language* (Alexander et al., 1977). Hence, for Alexander, pattern languages are both a theoretical account of the organisation of the built environment, and specific designed artefacts, whose purpose includes reinvigorating public participation in, and discussion of, architectural design. Our discussion of patterns reflects this and, unless explicitly stated, we are referring to constructed, documented patterns rather than patterns in the world.

In software engineering and HCI it is generally agreed that a pattern is a structured description of an invariant[2] solution to a recurrent problem in

2. It should be noted that the term *invariant* here refers to a set of shared characteristics of the recommended solution, but that the solution will need to be adapted to the specific circumstances in which it is applied. Hence, there is variability in the way that the solution is instantiated in individual applications, but the pattern describes the invariant core of solutions to the (recurrent) problem.

context, reflecting Alexander's problem-oriented approach. However, such an approach is not universal. A distinction can be drawn between design patterns, which centre on a problem and a proven solution, and activity patterns, which simply provide a description of existing patterns of activity (Bayle et al., 1998). For example, the patterns developed in the Pointer project (Martin et al., 2001; Martin et al., 2002), which seek to summarise findings from ethnographic studies, can be seen as "activity patterns" in Bayle et al.'s terms. Another area of work in software has proposed the idea of "AntiPatterns," which are examples of poor design practice together with descriptions of how the design could be repaired (Brown, Malveau, McCormick, & Mowbray, 1998). AntiPatterns have not attracted much attention within HCI, although there was some discussion at the CHI 2000 patterns workshop (Griffiths et al., 2000), in spite of many collections of examples of bad interaction design, with and without repairs. The validity of AntiPatterns in Alexandrian terms can be debated, because *patterns* are, by his definition, concerned with capturing *good* practice. However, their use in software is relatively common and they do occur in interaction design (see, e.g., Graham, 2003). Within this review, however, we concentrate on the predominant view (i.e., on "design patterns").

4.1. Characteristics of Pattern

A number of researchers have discussed what constitutes a design pattern and what distinguishes it from other design advice. Bayle et al. (1998) asserted that patterns are notable because they are based on examples, facilitate multiple levels of abstraction, bridge the gap between the physical and the social aspects of design and are amenable to piecemeal development. Fincher (1999) also identified capture of practice and abstraction as important, but adds organising principle to relate patterns to other patterns in a way that enables design; a value system that is embodied in the patterns; and a particular presentational style.

Perhaps the most comprehensive attempt to characterise patterns arises from the software engineering literature. Winn and Calder (2002) suggested nine essential characteristics of pattern, some of which reflect attributes also identified by previous researchers. Following we summarise the characteristics that Winn and Calder identify.

1. A pattern implies an artefact: A pattern should provide a higher-level picture of the shape of the artefact that it describes. The implication is that patterns must support the design of something.
2. A pattern bridges many levels of abstraction: A pattern provides design information at different levels of abstraction. Winn and Calder point

out that patterns in software engineering include both sample code and "big-picture" structure diagrams.

3. A pattern includes its rationale: It is both functional and nonfunctional. A pattern should include an explanation of why the solution is recommended, and what trade-offs are involved.
4. A pattern is manifest in a solution: It should be possible to see the pattern that has been used within the finished artefact, as a pattern relates to both design process and structure. Generally, it is not possible, by inspecting a piece of software, to identify whether a particular development process was used in its production. In contrast, if a software engineering pattern has been used, its structure will be evident in the code.
5. A pattern captures system hot spots: System hot spots are points within a software system that must be open to changes as the system evolves in response to a changing environment and modified requirements. By identifying invariants of good design, patterns also highlight design elements that must be open to change, and thus help to manage the interplay between stability and change.
6. A pattern is part of a language: Patterns are related to other patterns and work together to resolve the complexity of system design problems.
7. A pattern is validated by use: Patterns can only be proved through evidence of their existence in real artefacts and their contribution to design. Although this characteristic is similar to 4, there is a subtle distinction. Here the emphasis is on the evidence required to verify the existence of a pattern, which requires that the pattern is found in a range of successful system designs.
8. A pattern is grounded in a domain: Patterns relate to specific domains and have no meaning outside those domains. Patterns drawn from different domains should not be expected to work together, and discussion of patterns without consideration of the domains in which they are grounded is likely to be confused and confusing.
9. A pattern captures a big idea: Patterns should focus on key, difficult problems within a domain. Not every design problem warrants a pattern.

Within the field of HCI a number of other characteristics have also been debated. Bayle et al. (1998) raised several additional points.

10. Patterns support a "lingua franca": Patterns should support discussions with people who are not specialists in the domain. In contrast with the concerns of software engineering, patterns in HCI should be accessible and understandable by end-users.
11. Different patterns deal with problems at different scales: Some patterns in HCI deal with high-level issues such as business process or task struc-

ture, while others address low level details of graphical user interface construction such as the layout of tables.

12. Patterns reflect design values: Patterns are not neutral but explicitly reflect design values. The selection of patterns and the recording of patterns are value-laden activities, reflecting the priorities and motivations of the writer. We return to this point in more detail in section 7.
13. Patterns capture design practice: Patterns are derived from actual practice, not theoretical or conceptual proposals. This perspective relates to Winn and Calder's points 4 and 7 but here the emphasis is on the processes of identifying and developing patterns.

Figure 1 compares the position of Winn and Calder with that of a selection of authors in HCI who discuss the nature of design patterns. In this table we indicate a direct statement with a bullet and an implicit agreement (e.g., through the use made of patterns) with a question mark. We have included distinctions made by different authors, even where these are closely related. Figure 1 illustrates the level of debate on even the fundamental question of what constitutes a pattern and, to a degree, reflects a diversity of theoretical and philosophical perspectives on the nature of patterns. As we have already seen, Alexander viewed patterns both as a theoretical account of the built environment and as constructed artefacts to support design. Similarly, some authors are primarily interested in patterns as a way of capturing and sharing design knowledge and values, where it is assumed that documented patterns capture actual and observable successful design practice. Others see patterns primarily as an accessible form of design guidance and focus particularly on patterns that have immediate application and (in some cases) where that application can be automated.

Some of the requirements laid down by Winn and Calder have not been identified as important in HCI (e.g., the issue of system hot spots). Others, such as levels of abstraction within a pattern, are perhaps so obviously implied by the generic solution and concrete examples, as not to be stated explicitly by any HCI authors. Similarly, many HCI authors imply the focus on design of an artefact through inclusion of notions such as construction and generativity, although they do not mention this explicitly. It is clear, however, that there is a general agreement within HCI that patterns should allow communication between different groups; that pattern *languages,* as opposed to single patterns, are important; that patterns address problems at different levels; and that patterns involve questions of value.

4.2. Identifying Patterns

As we have seen, one of the distinguishing characteristics of patterns is that they are derived from practice rather than theory. In *The Timeless Way*

Figure 1. **A comparison of different perspectives on the *essential* characteristics of patterns. Used with permission of the author and the British HCI Group.**

Characteristic	Winn & Calder, 2002	Bayle et al., 1998	van Welie et al., 2000	Granlund et al., 2001	Borchers, 2000	Finlay et al., 2002	Fincher & Utting, 2002	Erickson, 2000a	van Duyne et al., 2003	Tidwell, 1998, 1999a
1. A pattern implies an artefact	•		?	?	•	?	?	?	•	?
2. A pattern bridges many levels of abstraction	•						?		?	
3. A pattern includes its rationale	•	?	•	?	•	?	?	?	•	?
4. A pattern is manifest in a solution	•									
5. A pattern captures system hot spots	•									
6. A pattern is part of a language	•		?	•	•	•	•	•	•	?
7. A pattern is validated by use	•	?	•	?		•		?		•
8. A pattern is grounded in a domain	•	?	?	•	•	?		•	•	
9. A pattern captures a big idea	•						•			
10. Patterns support a 'lingua franca'		•	?	?	•	•	•	•	?	•
11. Different patterns deal with problems at different 'scales'		•	•	•	•	?	?	•	•	•
12. Patterns reflect design values		•	?		•	•	•	•	?	•
13. Patterns capture design practice		•	•	?	?	?	•		•	?

of Building, Alexander (1979) described a process that begins by finding places that exhibit what he calls "the quality without a name," and then trying to identify the distinguishing characteristics that account for the success of the selected design solution. He then seeks to identify key "invariants" that are common to *all* good solutions to that design problem and not present in poor solutions.

In software engineering, it is usually agreed that patterns must be discovered by reference to design solutions, rather than being constructed from first principles. Coad (1992) suggested that "patterns are found by trial and error and by observation" (p. 153). Coad & Mayfield (1992) discussed "discovering" patterns from experience. Gabriel (1996b) and Meszaros (1996) both used the metaphor of "mining" patterns from existing designs. The mining metaphor has been used in workshops on patterns in HCI (van Welie et al., 2002), and many of the patterns offered by Tidwell (1998, 1999a), van Welie (2002–2005) and Brighton Usability Group (2003) are clearly based on observations of common design solutions.

Pattern mining starts with identification of good practice. However, it is not enough simply to capture good HCI practice: pattern mining requires capture of practice that is both good and significant (Fincher & Utting, 2002). Patterns are not intended to state obvious solutions to trivial problems or to cover every possible design decision, but to capture "big ideas" (Winn & Calder, 2002). A pattern should capture insights about the design that can inform even an experienced designer; explaining not only *how* a problem can be solved but also *why* a design choice is appropriate to a particular context. Fincher (2000a) reflected that identifying patterns in HCI (i.e., attributing positive qualities of an artefact to particular facets of the design) may be complicated by the high levels of complexity and context dependence in interaction. For example, certain designs (and patterns) may be appropriate in one culture, but not in another (Hall, Lawson, & Minocha, 2003). Other design elements may be appropriate only in the context of a particular "genre." These problems are not unique to HCI, nor are they insurmountable. Alexander and colleagues (Alexander, 1979; King, 1993) suggested that different cultures will develop and extend their own architectural pattern languages. Hall et al. (2003) suggested incorporating statements relating to cultural setting within the "context" of individual patterns. Walldius (2001) showed how patterns can be used to describe particular "genres" of film. van Duyne et al. (2003) used the idea of "site genre" as an organising principle within their Web design pattern language.

One element that is perhaps unique to interaction design patterns is the need to include the notion of temporality (Barfield et al., 1994; Borchers, 2001a). Unlike architecture, HCI deals with an artefact where time is significant and the contexts of, and solutions to, interaction problems are liable to

be dynamic rather than static. A pattern must therefore be able to capture these temporal aspects. Tidwell's (1999a) pattern, Step-by-Step Instructions exemplifies this issue. The pattern addresses a context in which

> A user needs to perform a complex task, with limited time, knowledge, attention, or space. Alternatively, the nature of the task is step-by-step, and it's meaningless to show all the action possibilities at once.

The solution suggested for the pattern is:

> Walk the user through the task one step at a time, giving very clear instructions at each step. Use visual similarities in all the steps, e.g., typography and layout, to maintain a rhythm throughout the task; make each step a focal point, both visually and in the user's "attention space." If information is needed from the user, ask for it in simple terms and with brevity, by keeping it short, you can better maintain the user's sense of flow through the whole step-by-step process.

Tidwell illustrates this solution by a simple circle and arrow diagram that represents the temporal/sequential character of the steps. It is clear that Tidwell's pattern relies on an understanding of the diagram as a series of user interface states with navigation between them. The use of alternative media (e.g., video) has been suggested to illustrate interactive time-based solutions (Borchers, 2000a) but the fundamental issue of abstracting true interaction rather than simply snapshots of appearance or behaviour remains.

On the other hand, patterns should also embody a time*less* quality, presenting a solution that is applicable regardless of particular platform or current technology. This is arguably a weakness in many current interaction design patterns, which are strongly based on a particular and current user interface paradigm (e.g., graphical user interfaces). Bayle et al. (1998) suggested that patterns that address interaction issues at a "high level" of abstraction may be timeless, but that patterns that are closer to the detail of interaction design perhaps necessarily reflect current paradigms. Tidwell's (1998, 1999a) Common Ground language includes examples of both types. Go Back to a Safe Place is equally applicable to desktop systems, mobile phones, personal digital assistants (PDAs) and aircraft flight control systems. It is likely to be relevant in any interactive system devised in the future, whereas Stack of Working Surfaces very clearly reflects current window based interaction styles.

The lack of variety of good examples and the immaturity of our design field as compared to architecture may lead to weaker examples being used as the basis of patterns in HCI (Fincher, 2002). Many interaction design "patterns" can be criticized for identifying *common* rather than necessarily *good*

practice. We shall return to the discussion of "good" practice in Section 7 where we discuss the role of values in patterns.

4.3. The Presentation of Patterns

Fincher (1999) indicated that identifying good practice is the "least part of the achievement" in developing patterns. Bayle et al. (1998) noted that it is relatively easy to observe phenomena in the world but much more difficult to use these observations to develop and explicate good patterns. To be useful patterns must present an abstraction of good practice at a meaningful level of granularity. Formulations that are too abstract will be impractical in real design use; those that are too specific will be difficult to reuse in new scenarios. Fincher and Utting (2002) compared abstraction in patterns to good teaching practice: it should facilitate understanding of the principles embodied in specific examples, to identify what is important in the examples. Winn and Calder (2002) suggested that patterns should present knowledge at graduating levels of abstraction.

The focus on design patterns as a distinct form for design guidance has led to debates about the content and structure of patterns. In software engineering, a range of alternative formats appear in Beck and Cunningham (1987), Coad (1992), Beck (1994), Beck and Johnson (1994), Gamma et al. (1995), and Fowler (1997). Meszaros and Doble (1998) presented a pattern language for pattern writing, suggesting a degree of stabilization around certain formats. Sharp, Manns, and Eckstein (2003) reported on the way that the format of patterns to support computer science education had to be modified to better suit the needs of their target audience.

In HCI, alternative formats have been followed by Tidwell (1998, 1999a), Borchers (2001a), van Welie, van der Veer, and Eliëns (2000), Martin et al. (2001, 2002), van Duyne et al. (2003), and Tidwell (2003). Some of these (e.g., Borchers, 2001a) reflect the layout and typesetting of *A Pattern Language,* for example, using bold fonts to highlight the key sections of the "problem" and "solution" and separating the text that describes the pattern's position in the language from the body of the pattern by using three diamonds. Others (e.g., Tidwell, 1998, 1999a) reflect the style of Gamma et al. (1995) with a series of specific headings. In Tidwell's case the headings used are Examples, Context, Problem, Forces, Solution, Resulting Context and Notes. Still others represent departures from previous forms (e.g., Martin et al. 2001, 2002; Tidwell, 2003; van Duyne et al., 2003). Representative examples of interaction design pattern forms have been collected in the Pattern Gallery (Fincher, 2000b). Several attempts have been made to identify common elements and to formalise these in some way, for example Griffiths et al. (1999) and the pattern language markup language PLML developed at the CHI 2003 workshop (Fincher, 2003). The

Document Type Definition (DTD) for PLML is given in Figure 3 and several collections have now been made PLML compliant, including van Welie (2002–2005).

Dearden et al. (2002a, 2002b) and Finlay et al. (2002) highlighted the degree to which different formats, including abbreviated patterns, affect the use of patterns in practical design settings.

4.4. Patterns, Guidelines, and Claims

Advocates of patterns in HCI have often sought to demonstrate clear distinctions between patterns and other forms of design guidance. For example, Borchers (2001a) suggested that patterns improve on style guides, guidelines, and standards:

> Through their structured inclusion of existing examples and insightful explanation not only of the solution, but also of the problem context in which this solution can be used, and the structured way in which patterns are integrated into the hierarchy of the language. (p. 60)

Patterns should also be compared to other efforts to reuse design knowledge, such as "claims"[3] (Sutcliffe, 2000; Sutcliffe & Carroll, 1999). To examine such arguments, we need to clarify both the forms of design guidance being discussed, and the contrasts identified. The following common types of design guidance can be distinguished:

1. Style guides, which are specific to an environment or product grouping (e.g., GNOME project, 2003; Microsoft Corporation, 2003).
2. General guidelines applicable to a range of systems (e.g., Smith & Mosier, 1986).
3. Standards, which may resemble guidelines, but carry some formal authority (e.g., International Standards Organisation [ISO], n.d.).
4. Claims, which incorporate both theoretical argumentation and specific illustrative examples (Sutcliffe, 2001; Sutcliffe & Carroll, 1999);
5. Heuristics, which are general statements of desirable properties (e.g., Nielsen, 1994).

---

3. http://www.co.umist.ac.uk/hci_design/appc.htm offers one approach to presenting claims. http://ucs.ist.psu.edu can be searched for examples of claims in the context of various projects (e.g., http://ucs.ist.psu.edu/dbitemview.asp?id=43&section=\Garden-com\Activity+Design\Rationale)

Figure 3. **DTD showing structure of pattern in PLML (Fincher, 2003)**

```
PLML v1.1
<!ELEMENT pattern (name?, alias*, illustration?, problem?, context?, forces?, solution?,
   synopsis?, diagram?, evidence?, confidence?, literature?, implementation?, related-patterns?,
   pattern-link*, management?)>
<!ATTLIST pattern patternID CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT alias (#PCDATA)>
<!ELEMENT illustration ANY>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT context ANY>
<!ELEMENT forces ANY>
<!ELEMENT solution ANY>
<!ELEMENT synopsis (#PCDATA)>
<!ELEMENT diagram ANY>
<!ELEMENT evidence (example*, rationale?)>
<!ELEMENT example ANY>
<!ELEMENT rationale ANY>
<!ELEMENT confidence (#PCDATA)>
<!ELEMENT literature ANY>
<!ELEMENT implementation ANY>
<!ELEMENT related-patterns ANY>
<!ELEMENT pattern-link EMPTY>
<!ATTLIST pattern-link
   type CDATA #REQUIRED
   patternID CDATA #REQUIRED
   collection CDATA #REQUIRED
   label CDATA #REQUIRED>
<!ELEMENT management (author?, credits?, creation-date?, last-modified?, revision-number?)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT creation-date (#PCDATA)>
<!ELEMENT last-modified (#PCDATA)>
<!ELEMENT revision-number (#PCDATA)>
```

A number of different aspects of patterns and pattern languages are suggested as distinctive. The major contrasts noted by van Welie et al. (2000), Borchers (2001a), Fincher (2000a), and Brighton Usability Group (2003) are as follows:

1. The level of abstraction at which guidance is offered.
2. The grounding of patterns in existing design examples, or "capture of practice."
3. The statement of the problem addressed by a pattern.

4. The discussion of the context in which a pattern should be applied.
5. The provision of a supporting rationale for the pattern.
6. The organisation of patterns into pattern languages.
7. The embedding of ethics or values in the selection and organisation of patterns.

To simplify discussion we note that standards are not a distinct form of guidance, but are distinguished by their authority. Indeed, the most commonly used standard in HCI (ISO, n.d.) includes many sections presented as guidelines (referred to as "principles" or "recommendations" within the standard). This leaves four distinct forms of guidance. Hence, we can identify 28 (4×7) distinct assertions. For example "interaction design patterns differ from heuristics because patterns are grounded in concrete examples." Examining these assertions it is clear that patterns differ from both style guides (because patterns aim to generalise away from particular implementation environments and from fine detail of user-interface rendering, and patterns discuss the context in which they are applicable), and from heuristics (because patterns identify particular solutions, the context of application, and are supported by a rationale). However, it is more difficult to distinguish patterns from guidelines (e.g., ISO, n.d.; Smith & Mosier, 1986) and claims (Sutcliffe, 2001; Sutcliffe & Carroll, 1999).

The following similarities and contrasts can be identified:

1. Level of abstraction: Patterns, guidelines, and claims can all be stated at various levels of abstraction. Some patterns tackle issues at a similar level of detail to typical examples of guidelines. For example, The Shield (van Welie et al., 2000) is comparable with ISO 9241-10 principle 3.3 (ISO, n.d.). However, the organisation of guidelines around particular styles of interaction (e.g., "data entry," "form filling," or "menu selection") may lead toward guidelines dealing with fine details of interaction (e.g., the arrangement of options within menus). In contrast, interaction design patterns can address larger scale issues over extended interactions. For examples, see Step-by-Step Instructions (Tidwell, 1998, 1999a), Easy Handover (Borchers, 2001a), or Recommendation Community (van Duyne et al., 2003). Claims can also describe such larger scale design issues.
2. Use of examples: Patterns, guidelines, and claims all include examples, but whereas examples in guidelines are usually phrased in general terms (e.g., "imagine an application that …"; Smith & Mosier, 1986), patterns and claims refer to specific implemented systems. There is a slight difference between patterns and claims in the use of examples. Patterns emphasise their grounding in multiple examples of successful

designs, whereas claims emphasise grounding in theory. A theory "motivates" a claim (Sutcliffe & Carroll, 1999), and the claim "explains" the design of a single artefact. Sutcliffe (2000) suggested that a pattern may be a "generic design for" a claim (p. 205).

3. Statement of the problem: Neither guidelines nor claims include a specific *problem* that they attempt to address.
4. Context: Some guidelines include "exceptions" to identify situations where they should not be applied, but this is not required in all cases. Claims include a specific scenario in which a particular artefact is used, which indicates a "context" in which the claim appears valid. In contrast, patterns aim to characterise a set of possible contexts in which the particular design advice should be followed. Hence the "context" in a pattern may generalise over the "context" for individual claims.
5. Supporting rationale: Guidelines, claims and patterns all provide some supporting rationale based in both primary research and other literature. The presentation of that rationale is more concise in Smith and Mosier's guidelines than is the case with typical patterns (e.g., Borchers, 2001a; van Welie et al., 2000). ISO 9241 (ISO, n.d.) does not include the references to the literature within the individual guidelines, instead providing a general bibliography.
6. Connections between elements: Cross-referencing is common to guidelines, claims, and patterns. However, while guidelines include occasional cross-referencing, both patterns and claims emphasize organisation and interdependence. We return to this issue in the next section.
7. Embedding values: At one level, guidelines, claims, and patterns all embody design values. However in guidelines and claims these values are implicit, patterns aim to make these explicit (Bayle et al., 1998), both in the detail of individual patterns and in the way that values inform pattern mining (Fincher & Utting, 2002).

In summary, patterns are potentially more general than existing examples of guidelines, use more specific examples, include the statement of a "problem" that they address, deliberately scope their context of application, and explicitly reflect particular design values. Patterns can be distinguished from claims by the inclusion of a problem statement, the requirement for multiple examples, the treatment of context, and the recognition that a pattern explicitly reflects selected design values. This comparison suggests that claims analysis might be a fruitful approach to the identification of patterns, but there may be a tension between the "theoretical and empirical" grounding of claims and the "value led" approach of patterns.

5. ISSUE 2: WHAT IS A PATTERN LANGUAGE?

Alexander's original work was not merely about individual patterns, but was explicitly concerned with the concept of pattern languages. Taken in isolation, patterns are, at best, "unrelated good ideas" (Alexander, 1996). However, combined in a language, patterns provide coherent support for design generation. In this section we examine what this means.

5.1. Pattern Languages and Pattern Catalogues

There are two forms of organisation readily evident in *A Pattern Language*. The patterns are collected into sets according to levels of physical scale (e.g., the first section of the language addresses the size and distribution of towns and cities, whereas later sections address smaller units such as neighbourhoods, clusters of houses, and individual rooms). In addition, the patterns form a network, where each pattern contains backward references to patterns that set its context (i.e., patterns that have already been used or selected) and forward references to patterns that can be used to help realise the current pattern. For example, the Street Café pattern begins by discussing patterns such as Identifiable Neighbourhood, Activity Nodes, and Small Public Squares that provide contexts to which a street café will contribute and ends by directing the reader to patterns that help realise the street café, such as creating an Opening to the Street, making the terrace double as A Place to Wait, and using Different Chairs. This directed network structure provides for Alexander's analogy with the production rules of a grammar (Alexander, 1979, p. 187).

In contrast, Gamma et al. (1995) described their efforts as a *catalogue* of patterns that have some interrelationships, but do not form a pattern language in Alexander's sense. Gamma et al. classify their patterns by their area of concern: creation of objects, structuring of software systems, or dynamic behaviour of systems. Other authors who have used classification schemes to organise pattern collections include Kendall, Murali Krishna, Pathak, and Suresh (1998), Roth (2002), Mahemoff and Johnston (1998), and Hussey and Mahemoff (1999). One of the early Object Oriented Programming Systems, Languages and Architectures (OOPSLA) workshops in which patterns were a major topic was concerned with creating a "handbook for software architects" (Anderson, 1993). Coplien and Schmidt (1995) discussed the distinction between pattern languages and catalogues, and suggested

> It is likely that catalogs of patterns … will provide the most payoff for pattern based software development over the next few years. It turns out that comprehensive pattern languages … are challenging to produce. (p. 322)

Gamma et al. (1995) expressed the hope that as more patterns are collected their catalogue might evolve and be organised into a language.

Some authors in software engineering have applied the concepts of refinement and specialisation to examine relationships between patterns (e.g., see Agerbo and Cornils, 1998; Mikkonnen, 1998; Tahara, Ohsuga, and Honiden, 1999; and Yacoub and Ammar, 1998). A similar approach for interaction design patterns is suggested by Mullet (2002), who proposes three possible relationships between patterns, namely derivation, where one pattern inherits elements from a higher level pattern; aggregation, where one pattern is contained within another pattern; and association, where one pattern uses another. Van Welie and van de Veer (2003) suggested a similar set of connections between patterns.

A number of pattern collections have been presented using a layered approach, with sets of patterns addressing different "levels" of a design problem. For example, Tahara et al. (1999) provided patterns addressing macro-architectural, micro-architectural, and finally object levels for the design of agent systems. Paternò (2000) suggested "task patterns" described in the ConcurTaskTrees notation, which are in turn linked to software "architectural patterns" that are described by configurations of reusable interaction components called "interactors." Granlund, Lafreniere, and Carr (2001) suggested interaction design patterns at the levels of "business domain," "business process," "task," "conceptual design," and "design."

5.2. The Organisation of Pattern Languages

While the majority of work in the PLoP conferences has been in the form of individual patterns or pattern collections, a number of networked languages have been presented. For examples see Johnson (1992), Richardson (2001), Hanmer (2000), Buschmann (2001), and Dyson and Anderson (1997). Networked pattern languages for interface software include Riehle and Zullighoven (1995), Bradac and Fletcher (1997), Towell (1998), Coldewey (1998), Judkins and Gill (2000), Marick (2000), and Berczuk, Appleton, and Cabrera (2000). Indeed, Beck and Cunningham's (1987) paper, which is generally accepted as the first application of patterns to software engineering, is a networked pattern language for the design of window-based applications. Richardson (2001) and Hanmer (2000) use an "enables" relationship between patterns, where later patterns enable the realisation of earlier patterns. Buschmann (2001) selected the term "completes" to express the relationship between patterns. This relationship in which one pattern "completes" another at a higher scale is evident in Alexander's writing, particularly in *A New Theory of Urban Design* (Alexander et al., 1987). Tidwell's (1998, 1999a) interaction design patterns are networked in a similar way. Borchers (2001a) provided

three examples of networked pattern languages for creating blues music, interaction design for multimedia exhibits and interface software design for multimedia exhibits. van Duyne et al. (2003) provided a networked "language" for the design of Web sites.

Fincher and others have drawn attention to the issue of finding a suitable "organising principle" for pattern languages in HCI (Fincher, 2002; Fincher & Utting, 2002; Fincher & Windsor, 2000). Fincher and Windsor (2000) suggested four requirements for an organising principle for a pattern language: it should provide a taxonomy to enable the user to find patterns; it should allow users to find related or proximal patterns; it should allow the user to evaluate the problem from different standpoints; and it should be generative, allowing users to develop new solutions. The two-stage organising principle that they propose focuses on the activities of design and the physical characteristics of interface elements rather than the activities of use. This focus is similar to that of other collections such as Tidwell (1998, 1999a). van Duyne et al. (2003) group their Web-design patterns to address different design aspects, beginning with "site genre," then examining issues such as "writing and managing content," and "making site search fast and relevant." Van Welie and van de Veer (2003) proposed a layered structure with patterns organised by posture, akin to van Duyne et al.'s genres; experience, relating to the particular expectation of the user in approaching the system; task, relating to sequences of interactions; and activity, relating to low level actions. The layers provide a mechanism for grouping the patterns but it is not clear how the relationships between the patterns are determined by it.

These structuring proposals all provide a way of taxonomising a pattern collection, but they do not actively support the process of identifying new patterns. The organisation is not *predictive*. Fincher (2002) contrasted this with other domains, notably chemistry, where the periodic table facilitated the discovery of previously unknown elements, because the organising structure illuminated "gaps" where these could fit. Fincher argues that the organisation of interaction design patterns by physical elements or common uses is arbitrary, whereas Alexander's patterns are organised by the "particular quality of the relationship between physical and psychosocial space" (p. 3). The former could be characterised as a structure, while the latter includes a clear structuring *principle*. Fincher (2002) suggested that Cognitive Dimensions (Blackwell & Green, 2003) might be a candidate for a structuring principle for interaction design patterns.

5.3. Notions of Generativity

A key concept in distinguishing pattern collections from pattern languages is the idea of generativity. In *The Timeless Way of Building,* Alexander explicitly

invokes comparison with generative grammars (Alexander, 1979, p. 187). One reading of the organisation of *A Pattern Language* (Alexander et al., 1977) suggests the idea of generating designs by implicit sequencing of decisions, derived by traversing the network of links between the individual patterns. This understanding is consistent with Alexander's description of case studies in *The Oregon Experiment* and *The Production of Houses* (Alexander et al., 1975, 1985).

In software engineering, a number of authors have sought to emulate this idea of a generative language. Beck and Cunningham (1987) suggested that a pattern language helps designers to ask and answer the right question at the right time (i.e., the language can be used to sequence design decisions). Beck (1994), Lea (1994) and Tahara, Toshiba, Ohsuga, and Honiden (2001) also suggest using the language for sequencing. The idea of patterns being connected by an enabling relationship, where later patterns enable the realization of earlier patterns is apparent in pattern languages in both software engineering and HCI (see, e.g., Aarsten, Brugali, & Menga, 1996; Dyson & Anderson, 1997). The notion of an "enables" or "completes" relationship between patterns (Buschmann, 2001; Hanmer, 2000; Richardson, 2001) is consistent with this reading of "generative" in the sense that a higher-level pattern implies the use of the lower-level patterns that enable it. In HCI Borchers (2001a) suggested this notion of generative sequencing of design decisions, which is also adopted by Finlay et al. (2002). Fincher and Windsor (2000) also reflect this by incorporating design process into their organising structure for pattern languages.

However, this is not the only way that the term *generative* has been discussed in software engineering and HCI. Gabriel (1996a) suggested that individual patterns can be considered "generative" because they give indirect advice about what to do to achieve a desirable outcome, rather than simply stating that the outcome is desirable. He gives the example of telling himself to "follow through" when hitting a tennis ball. This advice is indirect; it does not centre on the outcome of propelling the ball at speed. Instead it indicates a specific practical action that will achieve the desired result. Lea (1994) also emphasises this notion of generativity, as do Mahemoff and Johnston (1998).

Beck and Johnson (1994) suggested using patterns to construct a more complete design rationale for a whole system, analogous to a mathematical proof. In this analogy, patterns correspond to axioms (or theorems) of the design space. This approach is similar to Thimbleby's (1990) concept of "Generative Usability Engineering Principles," which specify constraints on permissible designs to ensure that resulting designs exhibit desirable properties. This may also be consistent with Alexander's analogy between pattern languages and Chomsky's grammars and with Alexander et al.'s (1987) approach in *A New Theory of Urban Design* and in *Notes on the Synthesis of*

Form (Alexander, 1964), both of which can be interpreted as forms of design by constraint solving.

Another concept of "generative" discussed in HCI, is the idea of generating an option space of alternative designs from which the design team should select (Dearden & Harrison, 1997; Lane, 1990; MacLean et al., 1991). Some pattern collections offer the reader a choice of alternative (incompatible) solutions to a design problem, from which one must be selected, based on specified attributes of the domain. For examples in software engineering see McKenney (1996), Dyson and Anderson (1997), Sandu (2001), Tahara et al. (1999, 2001), Mai and de Champlain (2001), Souza, Matwin, and Japkowicz (2002). In HCI an example is Tidwell's (1998, 1999a) alternative patterns Tiled Working Surface and Stack of Working Surfaces.

6. ISSUE 3: HOW ARE PATTERNS AND PATTERN LANGUAGES USED?

Alexander and colleagues provide four books in which they describe various experiments applying pattern-based design (Alexander et al., 1975; Alexander et al., 1985; Alexander, 1982; Alexander et al., 1987).

In the field of software engineering, although many patterns, pattern collections, and pattern languages have been published, there has been comparatively little discussion of the practical aspects of using patterns. Beck et al. (1996) reported on a panel discussion comparing experiences between various software organisations and Fraser, Beck, Booch, Johnson, and Opdyke (1997) debated whether frameworks and patterns actually reduce design costs. We have not found any published details of observational or empirical studies of software developers using patterns in practice.

Similarly, in HCI, there has been relatively little written about the practical details of using patterns in design projects (van Welie et al., 2000). Borchers (2001a) discussed how patterns might be applied at different stages of Nielsen's (1993) usability engineering lifecycle, and reports that patterns were used by various design teams in developing musical exhibits, but does not discuss precise details of the design activity. Windsor (2000) described using patterns to capture design rationale within specific projects. The Participatory Patterns Project (Dearden et al. 2002a, 2002b; Finlay et al., 2002) reported on simulated design exercises supported by patterns. Borchers (2002) reported on the use of patterns for teaching interaction design. Chung et al. (2004) described an empirical evaluation of the use of patterns for ubiquitous computing, which they (and we) believe to be the first controlled empirical study of the use of patterns with designers. In this section we consider these proposed uses of interaction design patterns in more detail.

6.1. Patterns for Participatory Design

Alexander argued that user participation in design is essential to successful building: "it is virtually impossible to get a building that is well adapted to these needs if the people who are the actual users do not design it" (Alexander et al., 1975, p. 42). His pattern language was intended to enable users to actively and directly design their own living and working spaces, in part by providing a common language with which they could make proposals and discuss ideas with an "architect-builder." An important practical element of this usage is the meaningful naming of patterns: in Alexander's language pattern names (without detail) are sufficient to facilitate this discussion. A similar emphasis on the need to develop a shared language is apparent in the participatory tradition in HCI (Ehn & Kyng, 1991; Ehn & Sjogren, 1991; O'Neill, 1998). King (1993) pointed out that a community using a pattern language in architecture is likely to evolve and develop their own specific pattern language or dialect.

Several authors in HCI have recognised this participatory focus. Bayle et al. (1998) highlighted participatory design as one possible application for pattern languages. Borchers (2001a) also mentioned participatory design as a possibility. The Participatory Patterns Project (Dearden et al. 2002a, 2002b; Finlay et al. 2002) investigated ways of combining pattern languages with other techniques for participatory interaction design, such as paper prototyping, and has found the approach promising.

A variation on the use of patterns in concert with paper prototyping is work by Lin and Landay (2002), who proposed to integrate patterns into a design sketching environment, allowing designers to drag and drop patterns into their sketches and customise them to meet local requirements. Although this approach is intended for experienced designers, its potential application within participatory design to support early prototyping with patterns is clear.

6.2. Patterns as Technical Lexicon

Many authors in software engineering report the use of pattern names as a specialist technical lexicon to support design debates. For example, Schmidt (1995) suggested that a knowledge of patterns "helped experts document, discuss and reason systematically about sophisticated architectural concepts" (p. 70). Cline (1996) suggested that patterns provide a "standard vocabulary" amongst developers. Meszaros (in Beck et al., 1996) stated a similar view. This standard vocabulary can also benefit design documentation, because a pattern name might be sufficient, in some contexts, to explain a complex design. Du and England (2001) proposed augmenting the User Action Notation

(Hartson, Siochi, & Hix, 1990) with references to patterns to produce more concise design specifications.

Cline (1996), Schmidt (1995), and Astrachan and Wallingford (1998) all suggested using patterns to educate novices about good software design, and integrating novices into design teams. Astrachan et al. (1998) claimed that patterns should form an essential part of the undergraduate computing science curricula. The explicit presentation of the content of patterns may also ease communication across development teams (Schmidt, 1995, p. 69). Goldfedder and Rising (1996) suggested using patterns to inform the review of a design and to aid documentation.

The use of patterns as an educational tool is carried through into HCI. One of the earliest HCI publications on patterns focuses on the use of patterns within an interaction design curriculum (Barfield et al., 1994). Borchers (2002) suggested two ways of using patterns within the curriculum: as a tool to present HCI design knowledge to students and as a methodology to support design. His experiences suggest that both can be successful and that students can grasp the patterns concept. Seffah (2003) and Sharp, Manns, and Eckstein (2003) took this a step further by suggesting the use of pedagogical patterns to design courses, as well as teaching interaction design and process patterns.

Cline (1996) advocated these ways of using patterns, but also suggested that patterns can be used proactively to suggest design structures. Where this proactive design generation is applied, Cline suggested that designers must apply a degree of "high-level pattern matching" (p. 47) to identify which patterns to use, and concludes that "the design patterns must be part of one's flesh and blood—looking things up in a book would be completely unacceptable in these on-the-fly situations" (p. 47). Goldfedder and Rising (1996) and Buschmann, Meunier, Rohnert, Sommerlad, and Stal (1996, p. 423) voiced a similar concern that the time to find a pattern increases as more patterns are published. This situation may suggest that designers will need to search a database of patterns to find one that matches their current problems, but whether this is a practical solution in the heat of real software development projects is open to debate. However, such resources are useful for students and practitioners seeking to enhance their knowledge and skills.

6.3. Patterns as Organisational Memory

In both HCI and software engineering, there has been some work on using patterns as part of an organisational memory. Beck et al. (1996) discussed efforts within specific organisations both to use patterns and to develop patterns that are specific to the domains in which those organisations operate. May and Taylor (2003) proposed patterns as a tool for organisational knowledge management. In HCI, Henniger (2001) suggested a process where each de-

velopment project begins by interrogating a corporate memory to retrieve and select patterns (and guidelines) to use within the project. Relevant patterns are identified by a rule-based system that matches patterns and guidelines against project characteristics (e.g., user populations, tasks, and GUI tools). The selected patterns are then passed to the project to consider. At the end of the project any patterns used are reviewed and may be updated based on the experience gained. Granlund et al. (2001) also suggested updating patterns on the basis of project experiences. Alexander et al.'s (1975) suggestions for the management of the pattern language in *The Oregon Experiment* (p. 136), part of which is an annual public review of the pattern language, can also be viewed as a form of organisational learning.

This context of organisational memory has led to the development of a number of tools to support the editing of patterns and pattern languages. Borchers (2001a, p. 195) describes requirements for PET, a "Pattern Editing Tool." Schuler (2002) and colleagues are developing an online pattern submission and discussion environment for recording patterns for "living communication." This environment allows participants to submit and edit their own patterns, and allows members of the public to review submissions.

Some authors have investigated incorporating software patterns into development tools, or implementing patterns as components of programming languages (see, e.g., Agerbo & Cornils, 1998; Chambers, Harrison, & Vlissides, 2000; Mapelsden, Hosking, & Grundy, 2002; Meijler, Demeyer, & Engel, 1997). This has also been proposed in interaction design (Lin & Landay, 2002; Molina, Torres, & Pastor, 2003). It can be objected that such efforts only incorporate the "solution" part of the pattern, but do not provide advice to software designers about when to use that particular pattern. Leacock, Malone, and Wheeler (2005) described the production of a library of interaction design patterns at Yahoo!, to which they hope to add visual assets and code fragments that can be reused by developers to produce systems that conform with the approved patterns.

6.4. Patterns as Lingua Franca

Crocker (in Beck et al., 1996) and Beck (1994) both discussed using patterns to support communication between designers responsible for the definition of the overall architecture of a system and designers responsible for applications software. Schmidt (1996) suggested using patterns to explain architectural design issues to managers. Fowler (1997) suggested using his patterns in collaboration with requirements analysts, clients and domain experts to develop specific models for particular projects.

In HCI, Erickson (2000a) also suggested patterns as a "lingua franca" to support and enhance communication about design, in particular advocating

the use of patterns to help users to engage with design processes. Granlund et al. (2001) suggested a design process of four phases: system definition, user profiling and task analysis, conceptual design, and "design." In each phase, patterns are used as archetypes to begin design discussions with users and clients. Borchers (2001a) reported on the use of three separate pattern languages, addressing different aspects of the design of a multimedia exhibit, namely designing and playing a piece of blues music, designing user interaction for the exhibit, and designing software to implement the exhibit's musical synthesis capabilities. Borchers suggested that, because the pattern format is familiar to designers from each of these different disciplines, they can more readily share their design thinking with each other across disciplinary boundaries. Martin et al. (2001, 2002) used patterns (although not design patterns) to present findings from ethnographic studies in a form that might be applied by software designers. Fernández, Holmer, Rubart, and Schummer (2002) expressed the hope that their patterns for groupware will improve communication within development teams, between development teams and end users, and between end users. Denning and Dargan suggested that a pattern language could provide "a method of mapping from human actions to software functions in a way that is intelligible to clients, designers and engineers simultaneously" (Denning & Dargan, 1996, p. 114). In the Participatory Patterns project, patterns are used to facilitate communication between users and website designers (Dearden et al., 2002; Finlay et al., 2002).

6.5. Patterns as Design Rationale

As we noted in the discussion of "generativity," there are a variety of understandings about the semantic relationships between patterns, pattern languages, and the designs produced from patterns. There is general agreement that patterns provide some rationale for particular design decisions, but the suggested (or implicit) structure of such rationales differs between authors.

Each of Alexander's patterns contains a discussion of the issues that surround the problem that the pattern addresses, and explains why the chosen solution is desirable. Cline (1996) argued that patterns can provide software engineers with design elements that have "well-understood trade-offs" (p. 47). Each of Gamma et al.'s (1995) patterns includes discussion of the trade-offs involved in selecting and using it. Additionally, within the "implementation" section of some of Gamma et al.'s patterns (e.g., Factory Method, State), alternative design options for certain aspects of the pattern are offered together with advice on selection.

Unlike Alexander's original work, some pattern languages in software engineering offer alternative patterns for similar problems, but designed for different contexts (e.g., Adams et al., 1996; Dyson & Anderson, 1997; Mai & de

Champlain, 2001; McKenney, 1996; Sandu, 2001). Fowler (1997) preferred to offer multiple ways of addressing a problem within a single pattern. Tahara et al. (2001) defined the context in which each of their patterns should be applied using a common set of indexing attributes. Souza et al. (2002) take a similar approach. Coplien (1998) used tables to relate the selection of certain patterns to analyses of commonalities and variabilities within a domain. This type of language suggests the possibility of using patterns in combination with a design rationale notation such as Questions, Options, and Criteria (QOC; MacLean et al., 1991). In such a combination, the patterns themselves could become reusable elements of the rationale.

Fowler (1997) suggests that his patterns can be used to suggest options for a design, which may be accepted, modified, or rejected. However, when the pattern is modified or rejected, the justification for that decision should be recorded as part of the design rationale. However, to date, we are unable to find any published demonstration of how such a design rationale would be constructed or presented.

Beck and Johnson's (1994) analogy with axiomatic mathematical proof suggests a more complete rationale connecting all of the design decisions. Such an interpretation would require a pattern language that is "generative" in the strict sense of a generative grammar, with the rationale for a design corresponding to a parse tree. Each of Beck and Johnson's patterns includes a "preconditions" section restricting the scope of the pattern (e.g., "you are writing a program that is animating a visual display in real time, probably in response to user input"; Beck & Johnson, 1994, p. 147). Hence, in this reading, the design rationale could be a proof that the pattern language (the set of axioms) entails the proposition that the specified context (or any context matching the required preconditions) implies the selection of the chosen design. This example highlights the fact that the context of a pattern is composed of two different parts. On the one hand, there is a context defined by the position of the pattern in the language (i.e., the larger patterns that it enables); on the other hand, part of the context refers to the nature of the environment in which the pattern is to be applied, the preconditions.

In HCI, different authors reflect these different understandings of design rationale. Pattern languages that make use of "enabling" links to generate designs are consistent with Beck and Johnson's (1994) idea of a proof (see Borchers, 2001a; Bradac & Fletcher, 1997; Coldewey, 1998; Dearden et al., 2002a, 2002b; Finlay et al., 2002; Judkins & Gill, 2000; Marick, 2000; Riehle & Zullighoven, 1995; Towell, 1998). However, these examples do not specify additional contextual details for each individual pattern. Rather the designer must make an initial decision about whether the language is relevant and, if so, the validity of the language and its correct application provides the rationale for the generated design. Tidwell (1998, 1999a) provides a generative language but

does include some patterns that represent distinct alternatives for similar problems (e.g., Tiled Working Surface and Stack of Working Surfaces). However, she does not specify in detail how to select between these options. Van Duyne et al. (2003) provided some alternative patterns (e.g., Fixed Width Screen Size and Variable Width Screen Size) together with textual discussion of suitable contexts for the application of each alternative, which would enable a form of rationale closer to Beck and Johnson's (1994) approach.

Pattern collections and catalogues (cf. Henniger, 2001; van Welie, 2002–2005) suggest a greater emphasis on pattern matching to construct the rationale. Granlund et al.'s (2001) approach also emphasises a rationale constructed by comparing pattern contexts with the conditions of a specific project. This approach is similar to Fowler's (1997). Mahemoff and Johnston (1998) and Hussey and Mahemoff (1999) begin with an analysis of relevant usability dimensions, which is similar to Tahara et al.'s (1999) approach, but they do not take this further into a defined process for using patterns.

Windsor (2000) reported on the use of patterns as an explicit mechanism for recording and organising the design rationale in an interaction design project.

7. ISSUE 4: VALUES AND PATTERN LANGUAGES

The idea of a "design language" is well-established in the sense of a collection of elements used to create a common design style (Rheinfrank & Eveson, 1996). However, Alexander's work clearly seeks more than just consistency of style. Rather, he chose the patterns in *A Pattern Language* to support a humane architecture that resulted in environments that he describes as "living" and "nurturing." In his keynote address to the annual conference on OOPSLA in 1996, Alexander (1996) draws attention to the "moral component" as central to his use of pattern languages in architecture. "In the architectural pattern language there is, at root, behind the whole thing, a constant preoccupation with the question, under what circumstances is the environment good?" (Alexander, 1996).

This leads to our fourth issue, the place of values in pattern languages for HCI. Issues of value are apparent in patterns in a number of different ways, including the following:

- The key properties that are examined when attempting to identify "good" design from which patterns may be discovered.
- The selection of, and the rationale provided for, individual patterns.
- The processes by which patterns are recorded and developed.
- The way in which patterns are used.

We examine these aspects in detail next.

7.1. The Properties Examined to Identify Patterns

Alexander discusses, at length, "The Quality Without a Name." He appeals to this "quality" to distinguish spaces and buildings that are "living" from negative or "dead" spaces. His patterns are then selected to enable the design of such "living" spaces. His procedure for identifying spaces with this "quality" is based on personal observation, but he claims that the "quality" is objective and empirical. To support this claim he reports that when people experience spaces that either do or do not have the quality, they exhibit a high level of agreement about its presence or absence. This might be interpreted as a claim of interrater reliability, though Alexander does not quantify the claim or provide any evidence. What is apparent is the holistic nature of the "quality" that Alexander is seeking. Dovey (1990) described Alexander's approach as implicitly phenomenological and suggests:

> The patterns are derived from the lived world (*lebenswelt*) of everyday experience and they gain their power, if at all, not by being proven empirically correct, but by showing us a direct connection between the pattern and our experience of the built environment. (p. 4; italics in original)

In Software Engineering, Gabriel (1996a) discussed the idea of code being "habitable" for those involved in the day-to-day maintenance of a system (pp. 9–16). He considers, and eventually rejects, "alive, whole, comfortable, free, exact, egoless and eternal" (p. 36) but admits "I still can't tell you what the quality is, but I can tell you some things about software that possesses it" (pp. 42–43). Gamma et al. (1995) and Cline (1996) emphasised designing software that is easy to reuse, in particular designing systems that are robust to certain types of change that may be necessary as requirements evolve. Winn and Calder (2002) described this as identifying system "hot spots" (i.e., distinguishing aspects of the system that should remain invariant from those that should permit change). Others highlight clarity of communication within development teams and between software development teams and maintenance teams (e.g., see Beck et al., 1996; Cline, 1996; Schmidt, 1995). Both Beck and Meszaros, in their contributions to Beck et al. (1996), described an aim of saving time in designing software, though Meszaros qualifies this by suggesting that patterns help "less experienced developers produce good designs faster" (p. 112). Tidwell (1999b) criticised software engineers for concentrating on such "technical" values, and for failing to apply values relating to users' experience of software.

The importance of values was recognised early in the development of patterns in HCI, for example Bayle et al. (1998) discussed this issue. Some authors have sought to identify an analogy for the "quality without a name" in

HCI. A definition developed at the 1999 ChiliPLoP workshop (Borchers 2000b; see also Borchers 2001a, p. 36) suggested "transparency"; Pemberton posited "engaging" (Pemberton, 2000); Van Welie et al. (2000) suggested that "usability" is sufficient; Christiansen (2005) suggested "competence affirmation"; Finlay et al. (2002) compared the "quality without a name" to Maslow's notion of "wholeness" (Maslow, 1970), which incorporates a sense of unity and integration as an essential component of self-actualisation. However, none of these proposals fully capture what Alexander intended where the "quality" is the essence of being alive.

It is not surprising that it is difficult to agree an appropriate analogy for the "quality without a name" given the holistic and experiential character of the "quality" described by Alexander. It is debatable whether the "quality" can exist at all in technology design, whether it should be sought or whether there are other properties (e.g., usability, acceptability, engagement) that may be more appropriate as outcomes of using interaction design patterns. Certainly, at the level of interface elements, usability may be more desirable. There is arguably also less agreement about what is fundamental to quality in interactive software compared with architecture, which may contribute to the difficulties in identifying an analogy to Alexander's "quality without a name." However, it is also valuable to explore what it might mean to be "living" in the context of technology design. Certainly properties of the living world, such as context awareness and adaptability, have been explored in technology design but, while seen as theoretically desirable, appropriate "technological" interpretations of these properties remain elusive. This concept is particularly pertinent when considering patterns at a more global level than those concerned with specific interface elements, for example, patterns that describe how, when and even if technology should be deployed. There are relatively few examples of patterns at this level, Schuler's (2002) patterns for "living communication" being the closest.

However, Fincher and Utting (2002) insisted that patterns and pattern languages must embody values since they advocate particular design ideas to be emulated. Hence, all pattern language development challenges practitioners and researchers in HCI to examine the value systems that they employ.

7.2. Values in the Selection of and Rationale for Individual Patterns

As well as informing the process of selecting "good" designs from which patterns might be identified, the individual patterns that are selected and the rationales provided within individual patterns help to make the authors' design values explicit. For example, Alexander includes patterns such as Old People Everywhere (40) and Four Storey Limit (21) that clearly reflect particu-

lar design values of integrated communities in touch with their environment. In HCI, patterns also reflect the values and priorities of their authors. For example, Borchers' patterns Attract-Engage-Deliver and Easy Handover both reflect the value of efficiency, in terms of the flow of people through the exhibition. In the case of the former this is from the perspective of the exhibition sponsor or organiser, wishing to maximise the number of people able to receive the message they wish to deliver. The latter is also concerned with efficiency but has a slightly different focus, reflecting the needs of the user within this rapid turnover. van Duyne et al. (2003) include a group of six patterns for "Building Trust and Credibility." These patterns focus on how designers can create Web designs to engender a sense of trust. However, the priority in these patterns is on establishing credibility through external appearance and explicit statements of trustworthiness rather than any attempt to address the actual behaviour (trustworthy or otherwise) of the organisation behind the site.

Values within pattern selection and rationale are reflected in the presentation of patterns at different levels, which provide a value-based context even where patterns cannot be used directly. Alexander includes patterns at a range of levels, from regional and whole city development, through local town planning, to individual neighbourhoods and buildings, to interior designs. Clearly not every potential user of the pattern language can exploit all of these patterns: home owners may only be able to use interior design patterns and some limited architectural patterns, whereas architects, builders, and town planners could utilise building and neighbourhood patterns directly. Relatively few stakeholders are in a position to make direct use of the highest-level patterns (e.g., Independent Regions), although Alexander would argue that each development contributes piecemeal to these global patterns. However, these patterns are also important in that they express the values that underpin the authors' view of architectural development, providing context for the lower level patterns. In HCI, there has been little work as yet on such high level, contextual patterns. Perhaps the most relevant work is the Public Sphere Project sponsored by Computer Professionals for Social Responsibility (Schuler, 2002). However it is easy to see parallels in terms of the types of environments, philosophies, and scales of development that many researchers and practitioners would wish to promote within interaction design.

7.3. Values in the Process of Developing Patterns

In *The Timeless Way of Building* Alexander (1979) described the evolution of pattern languages as a social process that is critically dependent on the involvement of users in using and discussing the language and the buildings generated by it (chapter 13). In particular Alexander suggests that professionalisation of debate about design leads people "lose confidence in

their own judgement" (p. 233) about what designs work for them. From *The Oregon Experiment*, it is apparent that Alexander et al. (1975) expected that specific communities will both adapt existing patterns to suit their needs and will create patterns and pattern languages that are specific to their situation. King (1993) also discussed the development of specific languages within specific communities. This view of the evolution of a pattern language as a social process might be compared with the concepts such as a speech community (Wynn & Novick, 1995), or a genre ecology (Erickson, 2000b).

In software engineering, a specific practice of "writers workshops" and "shepherding" has evolved to support the development of patterns and pattern languages. Each workshop has a "shepherd," who acts as chair and facilitator of the workshop and works with the authors of the papers to initially prepare the paper for the workshop. In the workshop, workshop participants discuss the paper but the author(s) are not allowed to comment. Their role is to listen to the discussion. After the workshop, the author(s) take the comments of the workshop into account in finalizing the paper for publication (Buschmann, Johnson, et al., 1996; Coplien, 2001; Kafura, Lavender, & Schmidt, 1995). A key value in this process is to ensure that the comments are always constructive, with the appointed "shepherd" taking responsibility for maintaining the constructive atmosphere.

There is some evidence of similar pattern writing workshops in HCI, for example, Borchers (2001a, p. 171) discusses how one pattern was modified in the course of such a workshop before final inclusion in the published pattern language and Schümmer et al. (2004) included shepherding activity. While writers' workshops emphasise pattern writing as a professional albeit apprenticed activity, the Participatory Patterns Project (Dearden et al., 2002a, 2002b; Finlay et al., 2002) reported that use of patterns in a participatory context permitted users to critique and make proposals for change to patterns. This suggests that Alexander's vision of users owning and evolving their own languages may be facilitated by participatory practices.

Another issue that has impacted the development of pattern languages, particularly in HCI, is the distribution of researchers interested in the subject, and the demands on researchers to publish and own work. Bayle et al. (1998) recognised that pattern language development needs to be a community effort, yet the competitive pressures within the wider research context can mediate against such a cooperative approach. This has led instead to competing voices and individual (and often repeated) efforts. Recent moves in developing a shared XML schema for patterns (Fincher, 2003) and the availability of Web-based communication systems to permit online collaboration in the effort of documenting and distributing pattern languages (e.g., van Welie, 2002–2005) are perhaps a move toward a more coherent sharing of the pattern development effort.

Schuler (2002) and colleagues are developing an online pattern submission and discussion environment for recording patterns for "living communication." This environment allows participants to submit and edit their own patterns, and allows members of the public to view currently submitted patterns. It is hoped that this environment will in future support a collaborative process whereby participants can select and develop the patterns toward a coherent pattern language.

7.4. Values in the Process of Using Patterns

Alexander's use of patterns to support participatory design is driven (in part) by a value system that treats localised control, and contextual sensitivity in design as essential. *The Linz Café* (Alexander, 1982) and *A New Theory of Urban Design* (Alexander et al., 1987) discuss the importance of making decisions on the actual construction site, and taking into account the surrounding context. In *The Oregon Experiment* and *The Production of Houses* Alexander et al. (1975, 1985) emphasised the use of patterns by a community to design for itself. In this situation it is important that the written patterns are not regarded as blueprints for design, rather they provide guidance that must be locally interpreted and must be open to challenge.

In the participatory tradition in HCI there is a similar commitment to users as active participants, rather than passive "subjects," and to the importance of local context in systems design. As discussed in section 6.1, The Participatory Patterns Project (Dearden et al,. 2002a, 2002b; Finlay et al. 2002) have conducted some initial investigations into this area. However initial results suggest that users may ascribe unwarranted authority to advice presented in the form of patterns (Dearden et al., 2002b). To avoid this, the authors advocate encouraging ownership and development of the language by users.

8. CONCLUSION: A RESEARCH AGENDA FOR PATTERNS IN HCI

In this article we have examined the patterns endeavour in HCI, looking in particular at the nature of patterns and pattern languages, the ways that patterns can be used, and the values they embody. From our review, it is clear that significant contributions have been made in the development of patterns and pattern languages which have been employed in the design of real systems (e.g., Borchers, 2001a; van Duyne et al. 2003). However, although the use of patterns is reported, there is little concrete evaluation of either the usefulness of pattern languages within the process or the contribution that they have made to the quality of the end product or to the design process (with no-

table exceptions, e.g., Borchers, 2002; Chung et al., 2004; Dearden et al. 2002a, 2002b; Finlay et al., 2002). Further, discussions of patterns and pattern languages so far within HCI have been dominated by form and examples, with limited examination of the philosophy and values of pattern based design. Given this context, we suggest that the research agenda for patterns in HCI should prioritise four areas, namely:

- Exploring appropriate ways to use pattern languages in design and in education, and evaluating the contribution that pattern languages can make.
- Finding ways to organise pattern languages in HCI so that patterns at different levels (from the broader social context of systems to the detail of interfaces) can be applied together in design.
- Exploring and improving the processes by which patterns are identified, recorded and reviewed so that the existing stock of patterns and pattern languages available in HCI can be constantly improved and enlarged, in particular to include generic patterns as well as those focused on particular platforms or interaction styles.
- Examining the way that values are explicated and promulgated in pattern languages and in pattern-led design.

We examine each of these areas in more detail.

8.1 Exploring Pattern Languages in Use

One of the most obvious weaknesses in HCI research on patterns to date is the lack of substantive evidence of their benefits for actual design practice. Perhaps understandably attention has focused on generating patterns, rather than on using them, and most researchers have developed their own languages for a variety of reasons. Significant effort is now required to examine the use of these languages in actual design (e.g., via empirical and observational studies) and in education to demonstrate what benefits might be gained from a patterns approach.

Three notable studies have demonstrated possible approaches to evaluating pattern languages in use. Borchers (2002) described evaluations of the value of patterns to student learning in two undergraduate HCI modules. Dearden et al. (2002) and Finlay et al. (2002) described a qualitative study of the role of patterns in simulated participatory design activities. Chung et al. (2004) reported on a structured empirical study using a pattern language in a simulated design activity including a group of experienced designers. These studies provide some possible ways in which such evaluations could be conducted. However, none of these studies can be treated as conclusive, so there is considerable need for further work.

Two key limitations of the work to date are as follows:

- The studies have not attempted to compare patterns with any other type of design advice, whereas a key claim made for HCI patterns is that they are in some way superior to other forms of design guidance.
- The studies have only investigated simulated design activities rather than longitudinal observations of "real world" developments, which may reveal different characteristics.

Given that a range of substantial pattern collections and pattern languages are now available, future research efforts must be focused on exploring how they can actually be used both in design and in education and in evaluating their effectiveness in these areas. Comparative and longitudinal studies in particular are needed.

A secondary related area of research may consider alternative ways in which pattern languages can be used to develop and document the rationale for design decisions.

8.2. Organising Pattern Languages

Fincher and Windsor (2000) raised the question of how pattern languages in HCI should be organised. This issue will have a significant impact on the ways in which larger pattern languages might be applied as practical resources in design, as it determines how easy it is to locate core and related patterns and whether patterns can be used effectively to generate solutions to problems. The organisation of pattern languages in HCI is particularly problematic because of the wide range of different levels that may be addressed by patterns in HCI, from the broader social context in which an interactive system is used, to the low-level details of interaction. Unlike architectural patterns, where "scale" provides a useful organising principle, in HCI the problem is multidimensional. Scale is important, but designers also address problems in terms of technology, task, information, and time. Providing organisational structures and retrieval approaches that reflect these different conceptualisations is a challenge that requires further research. Another challenge is creating an HCI pattern language that is truly "generative." Looking outside the practice to related theories and principles, as proposed by Fincher (2002), merits further investigation.

A related issue is the management and maintenance of pattern languages, as the interconnected structure of a pattern language becomes more complex. Here software tools may be helpful but they should be focused not on automating the application of patterns (patterns provide design guidance not blueprints) but on the intelligent management, organisation, and retrieval of pat-

terns to support their use in design practice. Here, attention must be paid to the differences between the sequence and pacing of activities in architectural production and in interactive systems development.

8.3. Improving the Production of Pattern Languages

To date pattern development has been relatively ad hoc, based on designer experience and largely individual or small group efforts. Much of the effort has been on developing individual patterns, collections and pattern languages. A number of different pattern languages and collections have been developed using different formats and structures, resulting in the same essential patterns (as recurrent phenomena) being reproduced (as explicit texts) in these different formats. For example, the pattern Step by Step Instructions first proposed by Tidwell (1999) described the same phenomenon as van Welie's (2002–2005) Wizard and van Duyne et al.'s (2003) Process Funnel. However, while Step-by-Step Instructions is a relatively generic pattern, the Wizard pattern, in the choice of name alone, implies a particular style of solution, and the Process Funnel, a particular type of application. This duplication of effort is further complicated by the pressures in academia to produce publications and by issues of copyright.

New members of the HCI patterns community have been encouraged to engage in writers' workshops to practice the skills of identifying and writing patterns. For some workshops, a proposal for a (new) pattern or pattern language is required to gain admission. This approach, when combined with the pressures in academia to produce publications, has some drawbacks.

- First, it makes the development and completion of patterns expensive, because much of the work takes place at face-to-face meetings at international events.
- Second, it assigns a high value to producing new patterns and new pattern languages for presentation at such workshops, but provides few incentives for evaluating, critiquing, improving, and evolving existing languages.
- Third, it is unclear how such improvements to an existing language should be published, if a pattern is regarded as the copyright of the initial author(s), then it is not clear how improvements can be made, recognised and distributed.
- Fourth, these workshops may fail to engage with some key stakeholders in pattern language development, namely end-users and designers, who should be the primary beneficiaries.

Bayle et al. (1998) suggested that identifying and writing patterns needs to become a genuine community effort within HCI. The Pattern Language for

Living Communication project (Schuler, 2002) has attempted to develop a broad international community, working together in a shared workspace where patterns can be proposed, critiqued and edited online. However, the project is not yet in a position to claim that a successful process for developing pattern languages has been found.

This situation indicates that research is required to develop better ways to encourage the widest possible collaboration in pattern language development. This may require some way of recognising and rewarding efforts made to contribute to a pattern language, as well as suggesting a requirement for new tools for computer supported collaboration in pattern language development. A means for reporting experiences of using particular patterns and taking such experience reports into account in improving patterns is needed. Some concept of community ownership of a language may be necessary, with collective democratic governance of the content of such languages. Alexander et al. proposed a possible model for an architectural pattern language in *The Oregon Experiment* (1975). An alternative approach might explore the different licensing approaches of the open source software community as models to permit community development of pattern languages.

More fundamentally, the duplication of patterns highlights the conflict between pattern languages that are aiming in some sense to be generic and applicable to a range of situations, those that are specific to a particular platform or interaction style, and those that are targeted to a particular domain or application area. The tendency in the proliferation of HCI pattern languages has been toward patterns directed at particular interaction styles (e.g., desktop computing) across a range of domains, or pattern languages that are specific to particular domains. We believe that the research community also needs to develop patterns and pattern languages that are generic across platforms, styles and domains, that are in some sense "timeless." To achieve this we need to improve our understanding of successful design in HCI. Pattern mining depends fundamentally on identifying successful design, a process that we need to refine. Frameworks for analysing design to identify the elements that make it successful are needed. Here work in other areas, from traditional usability assessment and more recent work on understanding the nature of user experience (e.g., Wright & McCarthy, 2004) to observations from other design disciplines (e.g., Dorst, 2003) may be useful.

8.4. Examining our Values

Values need to be given more attention in HCI generally, as we recognise the wider social and ethical implications of the technologies we design (Light, Blandford, Cockton, Dearden, & Finlay, 2004; Wild, Dearden, Light, & Muller, 2005). We need to consider the values HCI practitioners and researchers

should be promoting and how this might be done. In terms of patterns we need to think further about whether there is an equivalent of "quality without a name" for HCI and, if so, what it might be. We need to address how we can identify patterns that are both timeless and culturally sensitive. Understanding the role of values in design may help us to recognise the values embodied in patterns. There are also value issues involved in the development and use of patterns where the need for recognition of contribution needs to be balanced with openness for use and further development. The patterns community may be able to learn here from practices relating to open source software.

One area that has largely been neglected to date has been the consideration of global HCI design patterns. Alexander's Pattern Language begins with patterns on a global scale (e.g., Independent Regions) which may have little practical meaning to someone building a single home but which serve both to make explicit the value context in which more specific patterns should be understood, and to inform decision makers. HCI patterns have so far understandably concentrated on the specifics of designing particular applications but say little about how technology should be deployed or about the wider context. This is an area where patterns can contribute to the values debate.

Patterns and pattern languages offer an approach to design with much potential. Research in these areas is now needed to ensure that this promise is fulfilled and that pattern language research makes an effective and lasting contribution to the practice and understanding of interaction design.

NOTES

Acknowledgments. We are grateful for the comments and the feedback we have received from many people whilst developing this article. In particular, we should like to thank Barbara McManus, Elizabeth Allgar and Sally Fincher for stimulating discussions and their comments on early drafts. We should also like to thank all the reviewers for their helpful comments, and particularly Ralph Johnson and Jack Carroll for their insights and encouragement.

Support. The authors wish to acknowledge the support of their respective institutions in enabling this work.

Authors' Present Addresses. Andy Dearden, Communication and Computing Research Centre, Sheffield Hallam University, Sheffield, S1 1WB, UK. Email: a.m.dearden@shu.ac.uk. Janet Finlay, Innovation North, Leeds Metropolitan University, Caedmon Hall, Headingley Campus, Leeds, LS6 3QS, UK. E-mail: j.finlay@leedsmet.ac.uk

HCI Editorial Record. First manuscript received February 26, 2004. Revision received March 29, 2005. Accepted by John M. Carroll. Final manuscript received October 22, 2005. — *Editor*

REFERENCES

Aarsten, A., Brugali, D., & Menga, G. (1996). Designing Concurrent and Distributed Control Systems. *Communications of the ACM, 39*(10), 50–58.

Adams, M., Coplien, J., Gamoke, R., Hanmer, R., Keeve, F., & Nicodemus K. (1996). Fault tolerant telecommunication system patterns. In J. M. Vlissides, J. O. Coplien, & N. L. Kerth (Eds.), *Pattern languages of program design* (Vol. 2, pp. 549–561). Reading, MA: Addison-Wesley.

Adams, S. (1995). Functionality ala carte. In J. Coplien & C. Schmidt (Eds.), *Pattern languages of program design* (pp. 1–8). Reading, MA: Addison-Wesley.

Agerbo, E., & Cornils, A. (1998). How to preserve the benefits of design patterns. In *Proceedings of OOPSLA '98, ACM SIGPLAN Notices 33*(10), 134–143.

Alexander, C. (1964). *Notes on the synthesis of form.* Cambridge, MA: Harvard University Press

Alexander, C. (1979). *The timeless way of building.* Oxford, UK. Oxford University Press.

Alexander, C. (1982). *The linz café/das kafe linz.* Oxford, UK. Oxford University Press.

Alexander, C. (1996). The origins of pattern theory, the future of the theory, and the generation of a living world. *Keynote address to the eleventh annual conference on object-oriented programming systems, languages, and applications.* October 6–10, San Jose, CA. Retrieved February 18, 2006 from http://www.patternlanguage.com/archive/ieee/ieeetext.htm

Alexander, C., Davis, H., Martinez, J., & Corner, D. (1985). *The production of houses.* Oxford, UK: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). *A pattern language.* Oxford, UK. Oxford University Press.

Alexander, C., Neis, H., Anninou, A., & King, I. (1987). *A new theory of urban design.* Oxford, UK. Oxford University Press.

Alexander, C., Silverstein, M., Angel, S., Ishikawa, S., & Abrams, D. (1975). *The Oregon experiment.* Oxford, UK. Oxford University Press.

Anderson, B. (1993). Addendum to the proceedings of OOPSLA '92. Workshop report: Toward an architecture handbook. *OOPS Messenger, 4*(2), 109–113.

Anderson, B., Coad, P., & Mayfield, M. (1994). Addendum to the proceedings of OOPSLA '93. Workshop report: Patterns: Building blocks for object oriented architectures. *OOPS Messenger, 5*(2), 107–109.

Astrachan, O., Berry, G., Cox, L., & Mitchener, G. (1998). Design patterns: An essential component of CS curricula. *ACM SIGSCE Bulletin, 30*(1), 153–160.

Astrachan, O., & Wallingford, E. (1998). *Loop patterns.* Paper presented at PLoP '98. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/ P60.pdf

Barfield, L., Van Burgsteden, W., Lanfermeijer, R., Mulder, B., Ossewold, J., Rijken, D., & Wenger, P. (1994). Education: Interaction design. *ACM SIGCHI Bulletin, 26*(3), 49–86.

Bayle, E., Bellamy, R., Casaday, G., Erickson, T., Fincher, S., Grinter, B., et al. (1998). Putting it all together: Towards a pattern language for interaction. *SIGCHI Bulletin*, *30*(1), 17–33.

Beck, K. (1994). Patterns and software development. *Dr Dobbs Journal*, *19*(2), 18–23.

Beck, K., Coplien, J. O., Crocker, R., Dominick, L., Meszaros, G., Paulisch, F., et al. (1996). Industrial experience with design patterns. *Proceedings of the 18th International Conference on Software Engineering (ICSE 18)*. IEEE Computer Society, 103–114.

Beck, K., & Cunningham, W. (1987). *Using pattern languages for object-oriented programs* (Technical Report No. CR-87–43). Retrieved February 18, 2006, from http://c2.com/doc/oopsla87.html

Beck, K., & Johnson, R. (1994). Patterns generate architectures. In *Proceedings of the OO programming 8th European conference* (pp. 139–149). Berlin: Springer.

Berczuk, S., Appleton, B., & Cabrera, R. (2000). *Getting ready to work: Patterns for a developer's workspace.* Paper presented at PLoP 2000. Retrieved February 18, 2006, from http://jerry.cs. uiuc.edu/~plop/plop2k/proceedings/Berczuk/ Berczuk.pdf

Bjork, S., & Holopainen, J. (2004). *Patterns in game design.* Hingham, MA: Charles River Media

Blackwell, A., & Green, T. (2003). Notational systems–The cognitive dimensions of notations framework. In J. M. Carroll (Ed.), *HCI models, theories and frameworks: Toward a multidisciplinary science* (pp. 103–134). San Francisco: Morgan Kaufmann.

Borchers, J. (2000a). Interaction design patterns: Twelve theses. Position paper presented at the *Workshop on Pattern Languages for Interaction Design, CHI 2000 Conference on Human Factors in Computing Systems*, April 2–3, The Hague, Netherlands. Retrieved February 18, 2006, from http://www.hcipatterns.org/tiki-download_file.php?fileId=22

Borchers, J. (2000b). CHI meets PLoP: An interaction patterns workshop. *SIGCHI Bulletin*, *32*(1), 9–12.

Borchers, J. (2001a). *A pattern approach to interaction design.* Chichester, UK: Wiley.

Borchers, J. (2001b). A pattern approach to interaction design. *AI & Society*, *15*, 359–376.

Borchers J. (2002). Teaching HCI design patterns: Experience from two university courses. Position paper for *Patterns in Practice workshop at CHI 2002*, April 21–25, Minneapolis. Retrieved February 18, 2006, from http://www.hcipatterns.org/tiki-download_file.php?fileId=19

Borchers, J. O., & Thomas, J. C. (2001). Patterns: what's in it for HCI? Panel Discussion. *CHI '01 extended abstracts on human factors in computer systems* (pp. 225–226). New York: ACM Press.

Bradac, M., & Fletcher, B. (1997). A pattern language for developing form style windows. In R. C. Martin, D. Riehl, & F. Buschmann (Eds.), *Pattern languages of program design* (Vol. 3, pp. 347–393). Reading, MA: Addison-Wesley.

Brighton Usability Group. (2003). The Brighton Usability Patterns Collection. Retrieved February 18, 2006 from http://www.cmis.brighton.ac.uk/research/patterns/home.html

Brown, W. H., Malveau, R. C., McCormick, H. W., & Mowbray, T. J. (1998). *AntiPatterns: Refactoring software, architectures and projects in crisis.* New York: Wiley.

Buschmann, F. (2001). A pattern language for distributed object computing. Presented at *EuroPLoP 2001*. Retrieved February 18, 2006, from http://hillside.net/patterns/EuroPLoP2001/ papers/Buschmann.zip

Buschmann, F., Johnson, R., Coplien, J., Rising, L, Delano, D., Gamma, E., et al. (1996). *How to hold a writers' workshop*, Written in preparation for *PLoP 1996* Retrieved February 18, 2006, from http://www.cs.wustl.edu/~schmidt/writersworkshop.html

Buschmann, F., Meunier, R., Rohnert, H., Sommerlad, P., & Stal, M. (1996). *Pattern-oriented software architecture: a system of patterns*. New York: Wiley.

Chambers, C., Harrison, B., & Vlissides, J. (2000). A debate on language and tool support for design patterns. In *Proceedings of the 27th ACM SIGPLAN-SIGACT symposium on Principles of programming languages* (pp. 277–289). New York: ACM Press.

Christiansen, E. (2005). Competence affirmation as a complementary quality of human–computer interaction. To be presented at *Quality, Values and Choice a workshop at CHI 2005*, Portland, OR, 1st–7th April. Retrieved February 18, 2006, from http://www. bath.ac.uk/~maspjaw/workshops/QVC/Papers/Christiansen.pdf

Chung, E. S., Hong, J. I., Lin, J., Prabaker, M. K., Landay, J. A., & Lin, A. L. (2004). Development and evaluation of emerging design patterns for ubiquitous computing. In *Proceedings of DIS 2004* (pp. 233–242). New York: ACM Press.

Cline, M. P. (1996). The pros and cons of adopting and applying design patterns in the real world. *Communications of the ACM, 39*(10) 47–49.

Coad, P. (1992). Object-oriented patterns. *Communications of the ACM*, *35*(9), 152–159.

Coad, P., & Mayfield, M. (1993). Addendum to the proceedings of OOPSLA '92. Workshop Report: Patterns. *OOPS Messenger, 4*(2), 93–95.

Coldewey J. (1998). User interface software. Presented at *PLoP '98*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P13.pdf

Coplien, J. O. (1992). *Advanced C++ programming styles and idioms*. Reading, MA: Addison-Wesley.

Coplien, J. O. (1998). *Multi-paradigm design for C++*. Reading, MA: Addison-Wesley.

Coplien, J. O. (2001). *Writers workshop patterns*. Retrieved February 18, 2006, from http://c2.com/cgi/ wiki?WritersWorkshopPatterns

Coplien, J., & Schmidt, D. (1995). *Pattern languages of program design*. Reading MA: Addison-Wesley.

Dearden A, Finlay, J., Allgar, E., & McManus, B. (2002a). Evaluating patterns in participatory design. In *Adjunct Proceedings of CHI 2002* (pp. 664–665). New York: ACM Press.

Dearden A, Finlay, J., Allgar, E., & McManus, B. (2002b). Using pattern languages in participatory design. In Binder, T., Gregory, J., & Wagner, I. (Eds.), *Proceedings of PDC 2002*. Palo Alto, CA: CPSR.

Dearden, A. M., & Harrison, M. D. (1997). Abstract models for HCI. *International Journal of Human–Computer Studies, 46*(1), 151–177.

Denning, P., & Dargan, P., (1996). Action centred design. In Winograd, T. (Ed.), *Bringing design to software* (pp. 105–120). New York: ACM Press.

Dorst, K. (2003). *Understanding design: 150 Ways of looking at design.* Amsterdam: Book Industry Services.

Dovey, K. (1990). The pattern language and its enemies. *Design Studies, 11*(1), 3–9.

Du, M., & England, D. (2001). Temporal patterns for complex interaction design. In C. Johnson (Ed.), *Interactive systems: Design, specification, and verification, 8th international workshop, DSV-IS 2001, lecture notes in computer science 2220* (pp. 114–127). London: Springer.

Dyson, P., & Anderson, B. (1997). State patterns. In R. C. Martin, D. Riehl, & F. Buschmann (Eds.), *Pattern languages of program design* (Vol. 3, pp. 125–142). Reading, MA: Addison-Wesley.

Ehn, P., & Kyng, M., (1991). Cardboard computers: Mocking-it-up or hands-on the future. In J. Greenbaum & M. Kyng (Eds.), *Design at work* (pp. 169–196). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Ehn, P., & Sjogren, D. (1991). From system descriptions to scripts for action, In J. Greenbaum & M. Kyng (Eds.), *Design at work* (pp. 241–268). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Erickson, T. (2000a). Lingua francas for design: Sacred places and pattern languages. In *The Proceedings of DIS 2000* (pp. 357–368). August 17–19, Brooklyn. New York: ACM Press.

Erickson, T. (2000b). Making sense of computer-mediated communication (CMC): Conversations as genres, CMC systems as genre ecologies. In J. F. Nunamaker, Jr. and R. H. Sprague, Jr. (Eds.), *Proceedings of the thirty-third Hawaii international conference on systems science HICSS-33* (p. 3011). Piscataway, NJ: IEEE Press.

Fernández, A., Holmer, T., Rubart, J., & Schummer, T. (2002). Three groupware patterns from the activity awareness family. Presented at *EuroPLoP 2002.* Retrieved February 18, 2006, from http://hillside.net/patterns/EuroPLoP2002/papers/Fernandez_Holmer_ Rubart_Schuemmer.zip

Fincher, S. (1999). Analysis of design: An exploration of patterns and pattern languages for pedagogy. *Journal of Computers in Mathematics and Science Teaching: Special Issue on Computer Science Education, 18*(3), 331–348.

Fincher, S. (2000a). "Capture of practice": Is it obvious? *BCS HCI Group/IFIP WG 13.2 Workshop on HCI Patterns*, November 2000. Retrieved February 18, 2006, from http://www.cs.kent. ac. uk/people/staff/saf/patterns/bcs.pdf

Fincher, S. (2000b). The pattern gallery. Retrieved February 18, 2006, from http://www.cs.ukc.ac. uk/people/staff/saf/patterns/gallery.html

Fincher. S. (2002). Patterns for HCI and cognitive dimensions: Two halves of the same story? In J. Kuljis, L. Baldwin, & R. Scoble. (Eds.), *Proceedings of the fourteenth annual workshop of the psychology of programming interest group* (pp. 156–172). Brunel University, UK, June 2002. Retrieved February 18, 2006, from http://www.ppig.org/papers/14th-fincher. pdf

Fincher, S. (2003). *PLML: Pattern language markup language* Report of Workshop held at CHI September 2003, *Interfaces, 56* (pp. 26–28). Swindon, England: British HCI Group.

Fincher, S., Finlay, J., Greene, S., Jones, L., Matchen, P., Thomas, J. et al. (2003). Perspectives on HCI patterns: Concepts and tools, *CHI '03 extended abstracts on Human factors in computer systems* (pp. 1044–1045) April 05–10, Ft. Lauderdale, FL.

Fincher, S., & Utting, I. (2002). Pedagogical patterns, their place in the genre. In *Proceedings of ITiCSE.* June 24th–26th Aarhus, Denmark: ACM Press.

Fincher, S., & Windsor, P. (2000). Why patterns are not enough: some suggestions concerning an organising principle for patterns of UI design, *CHI'2000 Workshop on Pattern Languages for Interaction Design: Building Momentum.* Retrieved February 18, 2006, from http://www.cs.kent.ac.uk/people/staff/saf/patterns/chi00.pdf

Finlay, J., Allgar, E., Dearden, A., & McManus, B. (2002). Pattern languages in participatory design. In X. Faulkner, J. Finlay, & F. Detienne (Eds.), *People and computers XVI—Memorable yet Invisible, Proceedings of HCI2002* (pp. 159–174). London: Springer-Verlag.

Fowler, M. (1997). *Analysis patterns: Reusable object models.* Menlo Park, CA: Addison-Wesley.

Fraser, S., Beck, K., Booch, G., Johnson, R., & Opdyke, B. (1997). Beyond the hype: Do patterns and frameworks reduce discovery costs? *Proceedings of the 1997 ACM SIGPLAN conference on object-oriented programming systems, languages & applications (OOPSLA '97) SIGPLAN Notices, 32*(10) 342–344.

Gabriel R. (1996a). Introduction to *pattern languages of program design 2.* In J. M. Vlissides, J. O. Coplien, & N. L. Kerth (Eds.), *Pattern languages of program design* (Vol. 2). Reading, MA: Addison-Wesley.

Gabriel R. (1996b). *Patterns of software: Tales from the software community.* Oxford, UK: Oxford University Press.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1993). Design patterns: Abstraction and reuse of object-oriented design. In *Proceedings of the 7th European OO programming conference* ECOOP 93, LNCS 707 (pp. 406–431). Berlin, Germany: Springer.

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design patterns: Elements of reusable object-oriented software.* Reading, MA: Addison-Wesley.

Garlan, D., & Delisle, N. (1990). Formal specifications as reusable frameworks. In B. Bjorner, C. A. R. Hoare, & H. Langmaack. (Eds.), *VDM and Z: Formal methods in software development* (LNCS 428, pp. 150–163). New York: Springer-Verlag.

Garlan, D., & Notkin, D. (1991). Formalising design spaces: Implicit invocation mechanisms. In S. Prehn & W. J. Toetenel. (Eds.), *VDM '91: Formal software development methods* (LNCS 551, pp. 31–44). New York: Springer-Verlag.

Garlan, D., & Shaw, M. (1993). An introduction to software architecture. In V. Ambriola & G. Tortora. (Eds.), *Advances in software engineering and knowledge engineering,* Series on Software Engineering and Knowledge Engineering (Vol 2, pp. 1–39). Singapore: World Scientific Publishing Company.

GNOME Project. (2003). The GNOME human interface guidelines. Retrieved February 18, 2006, from http://developer.gnome.org/projects/gup/hig/1.0

Goldfedder, B., & Rising, L. (1996). A training experience with patterns. *Communications of the ACM, 39*(10), 60–64.

Grabow, S. (1983). *Christopher Alexander. The search for a new paradigm in architecture.* Stocksfield, Northumberland, UK: Oriel Press.

Graham, I. (2003). *A pattern language for web usability.* London: Addison-Wesley.

Granlund, A., & Lafreniere, D. (1999). A pattern-supported approach to the user interface design process. *Workshop report, UPA'99 Usability Professionals' Association Con-*

ference, June 29–July 2, Scottsdale, AZ. Retrieved February 18, 2006, from http://www.upassoc.org/ conf99reg/ws6.shtml

Granlund, A., Lafreniere, D., & Carr, D. A. (2001). PSA: A pattern supported approach to the user interface design process. In *Proceedings of HCI International 2001* (Vol. 1, pp. 282–286). Mahwah, NJ: Lawrence Erlbaum Associates, Inc.

Greenbaum, J., & Kyng, M. (Eds.). (1991). *Design at work* (pp. 169–196). Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

Griffiths, R. N., & Pemberton, L. (2001). Patterns in human–computer interaction design (panel session) In J. Vanderdonckt, A. Blandford, & A. Derycke (Eds.), *Proceedings of IHM-HCI 2001* (Vol. II). Toulouse, France: Cépaduès-Éditions.

Griffiths, R., Pemberton, L., & Borchers, J. (1999). Usability pattern language: Creating a community. In S. Brewster, A. Cawsey, & G. Cockton (Eds.), *Human–computer interaction–Interact 99* (Vol. II, p. 135). Swindon, England: British Computer Society. Outputs from the workshop retrieved February 18, 2006, from http://www.it.bton.ac.uk/staff/rng/UPLworkshop99

Griffiths, R., Pemberton, L., Borchers, J., & Stork, A. (2000). Pattern languages for interaction design: Building momentum. *CHI2000 Extended Abstracts* (p. 363). New York: ACM Press.

Hall, P. A. V., Lawson, C. J., & Minocha, S. (2003). Design patterns as a guide to the cultural localisation of Software. In *Proceedings of the 5th international workshop on internationalisation of products and systems* (pp. 79–88). 17th–19th July, Berlin.

Hanmer, R. S. (2000). Real time and resource overload language. Presented at *PLoP 2000.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/ plop2k/proceedings/ Hanmer/Hanmer.pdf

Harrison, N., Foote, B., & Rohnert, H. (Eds.) (1999). *Pattern Languages of Program Design 4.* Reading, MA: Addison-Wesley.

Hartson, H. R., Siochi, A. C., & Hix, D. (1990). The UAN: A user-oriented representation for direct manipulation. *ACM Trans. on Information Systems, 8*(3), 181–203.

Henninger, S. (2001). An organisational learning method for applying usability guidelines and patterns. In M. R. Little & L. Nigay. (Eds.), *Engineering human–computer interaction* (LNCS 2254, pp. 141–156). Berlin, Germany: Springer.

Hussey, A. (1999). Patterns for safer human–computer interfaces. In M. Felici, K. Kanoun, & A. Pasquini (Eds.), *Computer safety, reliability and security: SAFECOMP* '99 (pp. 103–112). Berlin, Germany: Springer-Verlag.

Hussey, A., & Mahemoff, M. (1999). Safety-critical usability: Pattern-based reuse of successful design concepts. In M. McNicol (Ed.), *4th Australian Workshop on Industrial Experience with Safety Critical Systems and Software (SCS) 99* (pp. 19–34). Canberra, Australia: ACS.

International Standards Organisation. (n.d.). ISO International Standard 9241 (Ergonomic requirements for office work with visual display terminals) Retrieved February 18, 2006, from http://www.iso.org

Johnson, R. (1992). Documenting frameworks using patterns. In *Proceedings of OOPSLA '92* (pp. 63–76). New York: ACM Press.

Johnson, R., & Cunningham, W. (1995). Introduction. In J. Coplien & D. Schmidt (Eds.), *Pattern languages of program design.* Reading, MA: Addison-Wesley.

Judkins, T. V., & Gill, C. D. G. (2000). Synthesizer a pattern language for designing digital modular synthesis software. Paper presented at *PLOP 2000.* Retrieved Febru-

ary 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Judkins/Judkins.pdf

Kafura, D., Lavender, G., & Schmidt, D. (1995). Workshop on design patterns for concurrent, parallel and distributed object oriented systems. In *Addendum to the proceedings of OOPSLA '95, OOPS Messenger, 6*(4), 128–131.

Kendall E. A., Murali Krishna, P. V., Pathak, C. V., & Suresh, C. B. (1998). Patterns of intelligent and mobile agents. In *Proceedings of the Second International Conference on Autonomous Agents*, 92–99. New York: ACM Press.

King, I. (1993, August). Christopher Alexander and contemporary architecture. *Special issue of Architecture and Urbanism.*

Laakso, S. (2003). User interface design patterns. Retrieved from http://www.cs.helsinki.fi/u/salaakso/patterns

Landay, J. A., & Borriello, G. (2003). Design patterns for ubiquitous computing. *Computer, 36*(8), 93–95.

Lane, T. G. (1990). *Studying software architecture through design spaces and rules* (Technical Report CMU/ SEI-90-TR-18). Carnegie Mellon University Software Engineering Institute.

Lea, D. (1994). Christopher Alexander: An introduction for object-oriented designers, *Software Engineering Notes, 19*(1) 39–46.

Leacock, M., Malone, E., & Wheeler C. (2005). *Implementing a pattern library in the real world: A Yahoo! case study.* Presented at the American Society for Information Science and Technology Information Architecture Summit, 3rd–7th March, Montréal, Québec, Canada. Retrieved February 18, 2006, from http://leacock.com/patterns

Lin, J., & Landay, J. A. (2002). Damask: A tool for early-stage design and prototyping of multi-device user interfaces. In *Proceedings of The 8th International Conference on Distributed Multimedia Systems 2002 International Workshop on Visual Computing* (pp. 573–580). September 26–28, San Francisco, CA.

Light, A., Blandford, A., Cockton, G., Dearden, A., & Finlay, J. (2004). Values in HCI: What drives our practice. In A. Dearden & L. Watts (Eds.), *Proceedings of HCI 2004: Volume 2* (pp. 211–212). Swindon, England: British HCI Group.

Maclean, A., Young, R. M., Bellotti, V. M. E., & Moran, T. P. (1991). Questions, options & criteria: Elements of design space analysis. *Human–Computer Interaction, 6*(3 & 4) 201–250.

Mahemoff, M. J., & Johnston, L. J. (1998). Principles for a usability-oriented pattern language. In P. Calder & B. Thomas. (Eds.), *OZCHI '98 Proceedings* (pp. 132–139). Los Alamitos, CA: IEEE Press.

Mai, Y., & de Champlain, M. (2001). A pattern language to visitors. Presented at *PLoP 2001.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ymai0/PLoP2001_ymai0_1.pdf

Mapelsden, D. Hosking, J., & Grundy, J. (2002). Design pattern modelling and instantiation using DPML. In J. Noble & J. Potter. (Eds.), *Proceedings of the 40th international conference on technology of object-oriented languages and systems TOOLS Pacific 2002* (pp. 3–11). New York: ACM Press.

Marick, B. (2000). Using ring buffer logging to help find bugs. Presented at *PLoP 2000.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2k/proceedings/Marick/Marick. pdf

Martin, D., Rodden, T., Rouncefield, M., Sommerville, I., & Viller, S. (2001). Finding patterns in the fieldwork. In W. Prinz, M. Jarke, Y. Rogers, K. Schmidt, & V. Wulf. (Eds.), *Proceedings of the Seventh European Conference on Computer Supported Cooperative Work* (pp. 39–58). Dordrecht, Netherlands: Kluwer.

Martin, D., Rouncefield, M., & Sommerville, I. (2002). Applying patterns of cooperative interaction to work (re)design: e-government and planning. In *Proceedings of CHI 2002* (pp. 235–242). New York: ACM Press.

Martin, R. C., Riehle, D., & Buschmann, F. (Eds.). (1997). *Pattern languages of program design 3.* Reading, MA: Addison-Wesley.

Maslow, A. (1970). *Motivation and personality* (3rd ed.). London: Harper & Row.

May, D., & Taylor, P. (2003). Knowledge management with patterns, *Communications of the ACM, 46*(7) 94–99.

McKenney, P. E. (1996, October). Selecting locking primitives for parallel programming. *Communications of the ACM, 39*(10), 75–82.

Meijler, T., D., Demeyer, S., & Engel, R. (1997). Making design patterns explicit in FACE: A framework adaptive composition environment. In M. Jazayeri & H. Schauer. (Eds.), *Proceedings of the 6th European Software Engineering Conference* (pp. 94–110). New York: Springer-Verlag.

Meszaros, G. (1996). Patterns for decision making in architectural design. In *Addendum to the proceedings of OOPSLA '95 OOPS Messenger, 6*(4), 132–137.

Meszaros, G., & Doble, J. (1998). A pattern language for pattern writing. In R. C. Martin, D. Riehl, & F. Buschmann (Eds.), *Pattern languages of program design* (Vol. 3, pp. 529–574). Reading, MA: Addison-Wesley.

Microsoft Corporation. (2003). *Windows XP visual guidelines.* Retrieved February 18, 2006, from http:// www.microsoft.com/hwdev/windowsxp/downloads

Mikkonen, T. (1998). Formalising design patterns. In *Proceedings of the 20th international conference on software engineering.* Los Alamitos, CA: IEEE Press.

Molina, P. J., Torres, I., & Pastor, O. (2003, February). *User interface patterns for object-oriented navigation upgrade IV,* 1. Retrieved February 18, 2006, from http://www.upgrade-cepis. org/issues/2003/1/up4=1Molina.pdf

Muller, M. J., Haslwanter, J. H., & Dayton, T. (1997). Participatory practices in the software lifecycle. In M. Helander, T. K. Laundauer, & P. Prabhu. (Eds.), *Handbook of human–computer interaction* (2nd ed.). Amsterdam: Elsevier Science.

Mullet, K. (2002). Structuring pattern languages to facilitate design. *CHI2002 Patterns in Practice: A Workshop for UI Designers* Retrieved February 18, 2006, from http://www.welie.com/patterns/ chi2002-workshop/Mullet.pdf

Nanard, M., Nanard, J., & Kahn, P. (1998). Pushing reuse in hypermedia design: Golden rules, design patterns and constructive templates. In *Proceedings of the ninth ACM Conference on Hypertext* (pp. 11–20).

Nielsen, J. (1993). *Usability engineering.* San Francisco: Morgan Kaufmann.

Nielsen, J. (1994). Heuristic evaluation. In J. Nielsen & R. L. Mack. (Eds.), *Usability inspection methods,* New York: Wiley.

Norman, D. (1988). *The psychology of everyday things.* New York: Basic Books.

Norman, D., & Draper, S. (Eds.). (1986). *User centered system design: New perspectives on human–computer interaction.* Hillsdale, NJ: Lawrence Erlbaum Associates, Inc.

O'Neill, E. (1998). *User-developer co-operation in software development. Building common ground and usable systems*, doctoral dissertation, Queen Mary & Westfield College, University of London.

Paternò, F. (2000). *Model-based design and evaluation of interactive applications*. Berlin: Springer-Verlag.

Pemberton, L. (2000). The promise of pattern languages for interaction design. Retrieved February 18, 2006, from http://www.it.bton.ac.uk/staff/lp22/HF2000.html

PLoP. (1998). *Proceedings of pattern languages of programs '98. Technical Report, WUCS-98–25*. Department of Computer Science, Washington University, WA. Retrieved February 18, 2006, from: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions

PLoP. (1999). *On-line proceedings of pattern languages of programs '99*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop99/proceedings

PLoP. (2000). *Proceedings of PLoP 2000*. Technical Report, WUCS-00–29, Department of Computer Science, Washington University, WA. Retrieved February 18, 2006, from http://jerry.cs. uiuc.edu/~plop/plop2k/proceedings/proceedings.html

PLoP. (2001). *On-line proceedings of the 8th conference on pattern languages of programs*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/ accepted-papers.html

PLoP. (2002). *On-line proceedings of the 9th conference on pattern languages of programs*. Retrieved from http://jerry.cs.uiuc.edu/~plop/plop2002/proceedings.html

PLoP. (2003). On-line proceedings of the 10th conference on pattern languages of program design. Retrieved February 29, 2006, from http://hillside.net/plop/plop2003/papers.html

Rheinfrank, J., & Evenson, S. (1996). Design languages. In T. Winograd. (Ed.), *Bringing design to software* (pp. 63–80). New York: ACM Press.

Richardson, C. (2001). A pattern language for J2EE web component development. Presented at *PLoP 2001*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/ crichardson0/PLoP2001_crichardson0_3.pdf

Riehle, D., & Zullighoven, H. (1995). A pattern language for tool construction and integration based on the tools & materials metaphor. In J. Coplien & D. Schmidt (Eds.), *Pattern languages of program design* (pp. 9–42). Reading, MA: addison-Wesley.

Rossi, G., Lyardet, F. D., & Schwabe, D. (1999). Developing hypermedia applications with methods and patterns. *ACM Computing Surveys*, *31*(4es) Electronic Symposium on Hypertext and Hypermedia, December 1999.

Rossi, G., Schwabe, D., & Garrido, A. (1997). Design reuse in hypermedia applications development. In *Proceedings of the eighth ACM conference on hypertext* (pp. 57–66). New York: ACM Press.

Rossi, G., Schwabe, D., & Lyardet, F. (2000). User interface patterns for hypermedia applications. In *Proceedings of the working conference on advanced visual interfaces* (pp. 136–142). New York: ACM Press.

Roth, J. (2002). Patterns of mobile interaction *Personal and Ubiquitous Computing, 6*(4) 282–289.

Sandu, D. (2001). Collection patterns. Presented at *PLoP 2001*. Retrieved February 18, 2006, from http:// jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/dsandu0/PLoP2001_dsandu0_1.pdf

Saunders, W. S. (2002, Winter/Spring). A pattern language. *Harvard Design Magazine, 16*, 74–78.

Schmidt, D. C. (1995). Using design patterns to develop reusable object-oriented communications software. *Communications of the ACM, 38*(10), 65–74.

Schmidt, D. C. (1996, December). Using design patterns to guide the development of reusable object-oriented software. *ACM Computing Surveys, 28*(4es).

Schmidt, D. C., Fayad, M., & Johnson, R. E. (1996). Editorial. *Communications of the ACM, 39*(10). *Special issue on Software Patterns*, 36–39.

Schuler, D. (2002). A pattern language for living communication. In T. Binder, J. Gregory, & I. Wagner (Eds.), *Proceedings of the participatory design conference 2002* (pp. 51–62). Palo Alto, CA: CPSR Press.

Schuler, D., & Namioka, A. (Eds.). (1993). *Participatory design: Principles and practice*. Hillsdale, NJ.

Schümmer, T., Borchers, J., Thomas, J., & Zdun, U. (2004). Human–computer-human interaction patterns: A workshop on the human role in HCI patterns. In *CHI '04 extended abstracts on human factors in computer systems* (CD). Retrieved February 18, 2006, from http://www.hcipatterns.org/CHI2004Workshop.html

Seffah, A. (2003). Learning the ropes: Human-centred design skills and patterns for software engineers' education. *Interactions, X*(5).

Sharp, H, Manns, M. L., & Eckstein, J. (2003). Evolving pedagogical patterns: The work of the pedagogical patterns project. *Computer Science Education, 13*(4), 315–330.

Smith, S. L., & Mosier, J. N. (1986). *Guidelines for designing user interface software*. Mitre Corporation Report MTR 9240, Mitre Corporation. Retrieved from http://hcibib.org/sam

Souza, J., Matwin, S., & Japkowicz, N. (2002). Evaluating data mining models: A pattern language. Presented at *PLoP 2002*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~ plop/plop2002/final/PLoP2002_jtsouza0_1.pdf

Sutcliffe, A. (2000). On the effective use and reuse of HCI knowledge. *ACM Transactions on Computer-Human Interaction, 7*(2), 197–221.

Sutcliffe, A., & Carroll, J. M. (1999). Designing claims for reuse in interactive systems design. *International Journal of Human–computer Studies, 50*(3), 213–241.

Tahara , Y., Ohsuga , A., & Honiden, S. (1999, May). Secure and efficient mobile agent application reuse using patterns. *Proceedings of the 21st International Conference on Software Engineering* (pp. 356–367). Piscataway, NJ: IEEE Computer Society Press.

Tahara , Y., Toshiba , N., Ohsuga, A., & Honiden, S. (2001). Agent system development method based on agent patterns. ACM SIGSOFT *Software Engineering Notes, 26*(3), 78–85.

Thimbleby, H. (1990). *User interface design*. New York: ACM Press.

Thomas, J. (2003). *A socio-technical pattern language proposal*. Retrieved February 18, 2006, from http:// www.truthtable.com/A_Sociotechnical_Pattern_Language.html

Tidwell, J. (1998). Interaction patterns. Presented at *PLoP 1998*. Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P29.pdf

Tidwell, J. (1999a). *Common ground: A pattern language for human–computer interface design.* Retrieved February 18, 2006, from http://www.mit.edu/~jtidwell/interaction_patterns.html

Tidwell, J. (1999b). *The gang of four are guilty.* Retrieved February 18, 2006, from http://www.mit.edu/~ jtidwell/gof_are_guilty.html

Tidwell, J. (2003). *UI patterns and techniques.* Retrieved February 18, 2006, from http://time-tripper.com/ uipatterns/index.php

Towell, D. (1998). Display maintenance. Presented at *PLoP 1998.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P15.pdf

van Duyne, D. K., Landay, J. A., & Hong, J. I. (2003). *The design of sites.* Boston, MA: Addison-Wesley.

van Welie, M. (2002–2005). *Interaction design patterns.* Retrieved February 18, 2006, from http://www. welie.com/patterns/index.html

van Welie, M., Mullet, M., & McInerney, M. (2002). Patterns in practice: A workshop for UI designers. In *CHI '02 extended abstracts on human factors in computer systems* (pp. 908–909). New York: ACM Press.

van Welie, M., & van der Veer, G. (2003). Pattern languages in interaction design: Structure and organisation. In *Proceedings of interact '03* (pp. 527–534). 1–5 September, Zürich, Switserland, Eds: Rauterberg, Menozzi, Wesson, Amsterdam, The Netherlands: IOS Press.

van Welie, M., van der Veer, G. C., Eliëns, A. (2000). Patterns as tools for user interface design. In C. Farenc & J. Vanderdonckt. (Eds.), *Tools for working with guidelines* (pp. 313–324). London: Springer-Verlag.

Vlissides, J. M., Coplien, J. O., & Kerth, N. L. (Eds.). (1996). *Pattern languages of program design 2.* Reading, MA: Addison-Wesley.

Wake W. C., Wake, B. D., & Fox, E. A. (1996). Improving responsiveness in interactive applications using queues. In Vlissides et al. (pp. 563 –573).

Walldius, Å. (2001). *Patterns of recollection: The documentary meets digital media.* Aura Förlag, Stockholm.

Weir, C., & Noble, J. (2003). A window in your pocket. In *Proceedings of the eighth European conference on pattern languages of program design* (EuroPLoP). Irsee. Universitäts Verlag Konstanz. Retrieved February 18, 2006, from http://www.charlesweir.com/papers/WindowInYouPocket.PDF

Weiss, M. (2001). *Patterns for e-commerce agent architectures: Using agents as delegates.* Paper presented at *PLoP 2001.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop2001/accepted_submissions/PLoP2001/mweiss0/PLoP2001_mweiss0_2.pdf

Wild, P., Dearden, A., Light, A., & Muller, M. (2005). Quality, values & choice in HCI. *Workshop at CHI 2005*, 4th–8th April, Portland OR.

Windsor, P. (2000). A project pattern language for user interface design. Presentation at *BCS HCI Group/IFIP WG 13.2 Workshop on HCI Patterns*, London, UK, November.

Winn, T & Calder, P. (2002, January/February). Is this a pattern? *IEEE Software, 19*(1), 59–66.

Wirfs-Brock, A., Vlissedes, J., Cunningham, W., Johnson, R., & Bollette, L. (1991). Designing reusable designs (panel session): Experiences designing object oriented

frameworks. In *Proceedings of OOPSLA / ECOOP 90, Addendum: systems, languages, and applications* (pp. 19–24). New York: ACM Press.

Wright, P., & McCarthy, J. (2004). *Technology as experience.* Cambridge, MA: MIT Press.

Wynn, E., & Novick, D. G. (1995). Conversational conventions and participation in cross-functional design teams. In *Proceedings of COOCS, 95* (pp. 250–257). New York: ACM Press.

Yacoub, S. M., & Ammar, H. H. (1998). A pattern language of statecharts. Presented at *PLoP 1998.* Retrieved February 18, 2006, from http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P22.pdf