

# The Timeless Way: Making Living Cooperative Buildings with Design Patterns

Lyn Pemberton and Richard N Griffiths

University of Brighton,  
Faculty of Information Technology,  
Watts Building, Lewes Rd., Brighton, UK  
Email: lyn.pemberton@brighton.ac.uk  
r.n.griffiths@brighton.ac.uk

**Abstract.** Interfaces to information systems, and the buildings in which such systems are embedded will typically be the result of the work of a large number of different disciplines, potentially ranging from ethnographers to architects. A common language and conceptual framework has the potential for greatly enhancing the effectiveness and ease of cross-disciplinary communication. In this paper we describe some aspects of the notion of design patterns developed by architect Christopher Alexander and colleagues in the 1970's. We briefly show how Alexander-style patterns can be used for analysis and design in some of the disciplines implicated in the creation of successful cooperative buildings — interface design, ergonomic design, functionality design and office design and suggest that pattern languages might be a way of bridging the communication gaps between professions to produce a shared vision of the cooperative building project.

**Keywords.** design patterns, guidelines, human-computer interface, ergonomics, interior design, software design, communication

## 1 Introduction

Interfaces to systems, and the buildings in which systems are embedded will typically be the result of the work of a large number of different disciplines. Systems analysts, knowledge engineers or even ethnographers may have been involved in discussing needs with sponsors, domain experts, managers and potential end-users. They will need to talk to software designers who will in turn communicate their ideas to programmers. In some cases the designers of the casing for the systems will need to consult furniture designers and interior designers, and these experts and others will in turn liaise with architects. Clearly it would make sense if, rather than each profession conceptualising the problem and the solution in its own way, they shared some terms and mental structures for communicating the design goals and constraints which apply in each area. Buildings need to be designed by people capable of speaking a common language.

We think the various processes involved in design can usefully be informed by the imaginative and thoughtful work on architectural and engineering design which took place from the sixties onwards in the US and Europe<sup>1</sup>. In particular, in the complex multidisciplinary domain of cooperative building design, we see a place for design patterns, originally introduced by architect Christopher Alexander and colleagues, in two books, *A Timeless Way of Building* (Alexander, 1979) and *A Pattern Language* (Alexander et al, 1977). Though developed in the domains of architecture, town planning and interior design, Alexander's thinking on design patterns has been applied very intensively over the last few years in one area of computer systems development, that is object-oriented design. There it has proved very powerful and fruitful (Gamma et al, 1995). This proof that the design pattern approach travels well has encouraged other initiatives in the software design area<sup>2</sup> and makes us optimistic that it should be a useful conceptual tool for the range of specialisms which would go to make up the design thinking for a co-operative building.

In this paper we first give a brief introduction to design patterns. We then give some examples of the patterns which can be uncovered at four levels: interface design, software functionality, ergonomic design and office design and conclude with some practical steps which would represent a way forward.

## 2 Background: What Are Design Patterns?

Patterns grew out of Alexander's disaffection with the quality of architecture in the 1960's, which he attributed in part to the misapplication of formal methods in architectural design. This had resulted in buildings which failed to fulfil the real needs of the people who lived and worked in them, which failed to adapt to local social and physical environments and which people simply did not like (cf. Lea, 1997). Alexander contrasts these modern failed building with the many successful, "living" buildings, created in other societies, buildings which for Alexander embodied "the quality without a name", a recognisable but indefinable quality which floats in the semantic space bordered by terms such as "alive", "whole", "comfortable", "free", "exact", "egoless" and "eternal". Patterns are conceptual tools for helping people design buildings which might themselves have that quality.

A pattern is the solution to a problem in a context. To put it less succinctly, in a context or a set of situations, a problem or clash of constraints will occur, which is amenable to resolution by a canonical design form or solution. The pattern encompasses all three elements: the situation, the problem of clashing constraints or forces,

---

<sup>1</sup>It is encouraging to see software design finally positioning itself as one design discipline among others and thus enabling itself to benefit from the work done by design thinkers such as Christopher Alexander, John Chris Jones, Nigel Cross, Bryan Lawson, Geoffrey Broadbent, Horst Rittel and Bruce Archer to name only some of the stars in the design firmament. For a good introduction see Cross 1984 and bibliography. For recent work in software design which takes a design research slant, see some of the contributions in Karat (1991) and particularly (Winograd 1997).

<sup>2</sup>In particular the Pattern Language workshop organised at CHI 97 (Erickson).

and the canonical solution. An example problem in a context, from Alexander, would occur where parents and children live in a house together (context), in which the parents would like their own space away from the children, but still want to be able to go easily to the children if, for instance, they are ill or anxious at night (problem). A standard solution would be to create a separate space for the parents, still within easy reach of the children's room. This is the pattern "Couple's Realm," number 136 among the 253 patterns presented in *A Pattern Language*. Like all the patterns it is connected both to larger patterns, which it completes, including, in this case, "house for a couple" and "intimacy gradient", and to smaller patterns which in turn complete it, in this case "low doors", "sitting circle", "light on two sides" and "marriage bed" among others.

At the highest level we find patterns such as "Independent regions", "Country towns" and "The distribution of towns", while at the other end of the scale are detailed patterns covering the need for a bench by a front door and for different sorts of chairs to be provided in a room. Together the linked patterns form a Pattern Language, a kind of informal grammar for buildings and spaces.

The solutions are not simply pre-formed parts of a building kit. A pattern is an abstraction from any specific examples: this is what gives patterns their generative power. They do not supply ready-made answers: people need to exercise their own creativity to implement a pattern. In addition, because they involve abstracting away from individual cases, patterns are hard to discover and may take a long time to describe adequately. (Alexander and colleagues spent over ten years refining *A Pattern Language* and Alexander commented that finding patterns was as hard as theoretical nuclear physics).

Alexander's own patterns are structured and formatted as follows (Alexander et al, 1977):

|                      |  |
|----------------------|--|
| <b>Title:</b>        | Which succinctly (and evocatively) captures the <i>solution</i> that the pattern offers.   |
| <b>Asterisks:</b>    | To mark the significance of the pattern, two asterisks marking a "true invariant", one marking a pattern which has made progress towards identifying such an invariant, but which needs further work, and no asterisks indicating confidence that an invariant has not been established, and that variations are to be expected. |
| <b>Picture:</b>      | "... which shows an archetypal example of that pattern." This may be literal or impressionistic. A subsidiary function may be to help the reader remember and find the pattern subsequently.   |
| <b>Introduction:</b> | Setting the context and linking to higher level patterns.  |
| ☐☐☐                  | To mark the beginning of the problem.  |
| <b>Headline:</b>     | (In bold type) summarising the essence of the problem.   |

|                      |  |
|----------------------|--|
| <b>Problem body:</b> | Describing the empirical background of the pattern, the evidence for its validity, range of variation of manifestation.  |
| <b>Solution:</b>     | (In bold type) Describing the “... field of physical and social relationships which are required to solve the stated problem in the stated context.” as a statement, in imperative form. |
| <b>Diagram:</b>      | Of the solution (For Alexander the solution should always be capable of a diagrammatic representation.)  |
| ☒☒☒                  | To mark the end of the main body of the pattern.   |
| <b>Connections</b>   | To lower level patterns which are required to complete this pattern.   |

Pattern makers in other disciplines have adapted this layout as needed.

### 3 Patterns in Cooperative Building Design

We think patterns should provide a valuable conceptual tool in several areas of design implicated in building design, from representations of the findings of ethnographic studies of the workplace to desk and console design to the nitty gritty of object-oriented programming, where they already enjoyed a great success. At all these levels they could enable designers to benefit from the knowledge and experience of creators of successful systems, providing reusable templates adapted to fit the particular issues which the designer is addressing. Above all, patterns, because they are themselves alive and engaging, provide a means of communicating either between designers of similar artefacts, e.g. one architect or interior designer to another, or designers looking at reshaping the environment at quite different levels, e.g. furniture designer to interface designer, or ethnographer to architect.

Why? What do software systems have to do with buildings? Pattern languages seem to resonate with three sorts of rhetoric heard in the more user-aware parts of the software design community:

- many designers recognise that there is a measure of discontent amongst users about the systems designed “for” them, and this is, in part at least, to be laid at the feet of a rigid application of formalistic methods of analysis and design which shut out the user from all except the very earliest design discussions.
- intuitively, the idea of “liveness” in buildings, the quest for the quality without a name, has much in common with the intuition that the systems which genuinely enable users to work as they want are those which we can call “engaging” or “convivial” systems. These are terms, like Kay’s (1990) and Tognazzini’s (1993) remarks on the place of magic in the interface, which seem to bound a

similar semantic space to the quality without a name in the sphere of software. Systems with this quality, like Goldilocks' porridge, are “just right”.

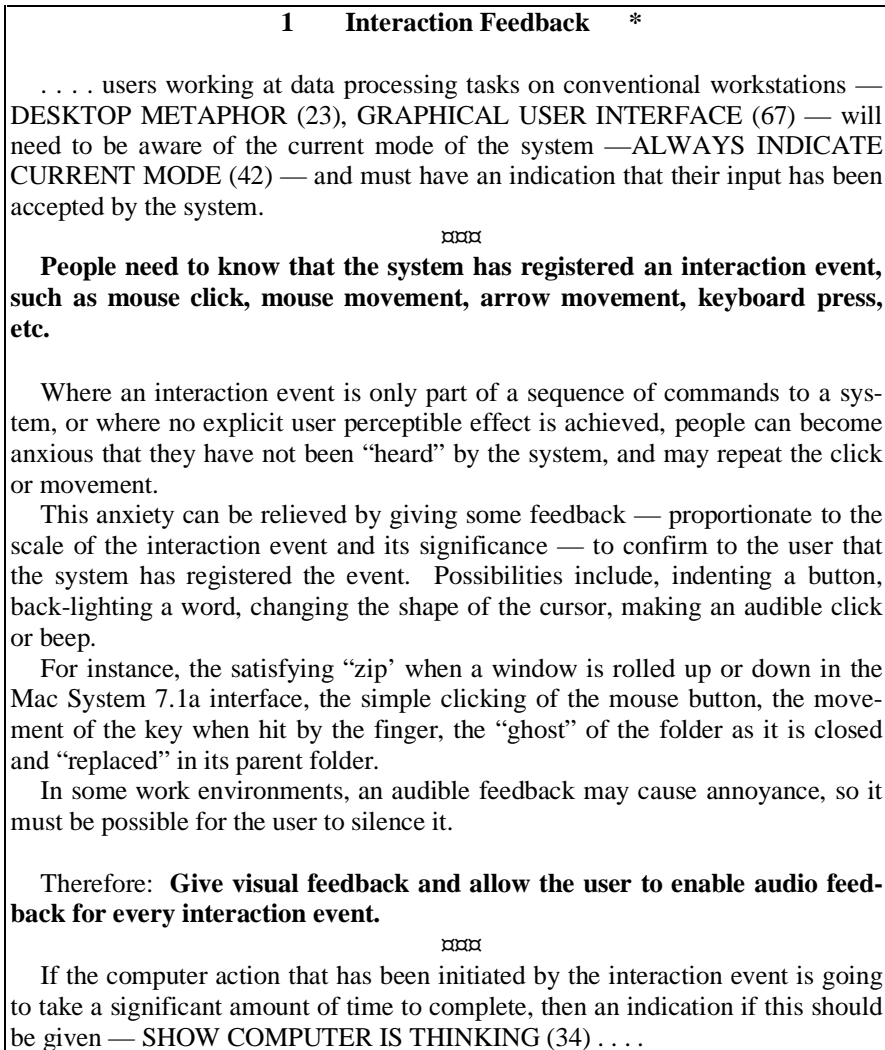
- much of Alexander's approach is evocative of recent moves in the HCI community towards participative design and user centredness. Alexander goes further than mere participation, in fact. The whole of "A Pattern Language", for instance, is predicated on the assumption that the person using the patterns will not be the trained architect but the person who will be living or working in the building.

### 3.1 Why are design patterns relevant to interface design?

Interface design is particularly obvious as a domain that would lend itself to pattern approaches. Interface design has always been about metaphors, however loosely conceived: even a primitive command line interface was based on a metaphor of conversation. With the advent of the graphical interface these metaphors have become increasing spatial. The desktop metaphor is ubiquitous, in the Web (itself a spatial metaphor) terms such as home and site are commonplace, and it is now common to speak of information spaces and to design visualisations of data in 2-D and even 3-D spaces. The mapping between source and target terms in any metaphor is never complete and one-to-one: metaphors, in interface design as in language, will always break down sooner or later (cf. Lakoff and Johnson, 1980). However, again in interface design as in language, users seem to be able to handle this: if not they would feel less easy about desktops which contained not only folders but also objects rarely encountered on top of desks such as windows, menus and wastebaskets. People have little difficulty in associating modern interfaces with the sorts of objects and relationships which are the stuff of Alexander's patterns.

#### 3.1.1 *Some Design Patterns for interface design*

When searching to identify design patterns in interfaces, it is probably as well to remind ourselves that whereas the buildings which Alexander considered were the result of many hundreds of years of development, interfaces are at a relatively early stage. There are few universally acclaimed solutions and this may mean that it will be easier to identify problems than to describe the solutions which address them. In Alexander's terms, it may be that any patterns we do find will not deserve asterisks. However, in a pioneering spirit, below are just a few patterns which can be discerned in interface design. The first of these, which describes the need for the user to receive feedback when interacting with a system, is described in full, in Fig. 1, while the rest are left in outline "context, problem and solution" form. Figures in parentheses are references to other, fictional, patterns which would have to exist in a complete pattern language.



**Fig. 1.** Pattern describing the need for the user to receive feedback when interacting with a system

An immediate reaction to the idea presented here may be that it simply replicates (in a rather long winded way) what has already been done in HCI design through the introduction of style guides such as the Macintosh Human Interface Guidelines (Apple, 1992) — which actually references Alexander’s work, The Windows Interface: An Application Design Guide (Microsoft, 1987), NASA Human-Computer Interface Guidelines (Carlow, 1992), and *very* many others. However we claim that what is missing from these style guides, and is added in the pattern approach, is the explicit acknowledgement that whole *patterns* are being recorded. This implies that the meta-

information surrounding the simple imperative instruction<sup>3</sup> normally found in guidelines is recorded — and must have been thought through. This means explicitly setting the context in a hierarchical structure of patterns, drawing attention to the *problem* that the pattern solves, conceiving of the solution to the problem as resolving conflicts in a field of interacting social and physical relationships, and considering and stating degrees of confidence in the invariance of the pattern. This makes patterns an altogether richer resource for the designer than lists of guidelines — more akin to the resources to be found in what we have called “craft wisdom” books, a good example of which is Cooper (1995) — but expressed in a canonical form.

Secondly, the use of patterns implies an emphasis on the *process* of developing and using guidelines, rather than on the product — a list of imperative directions. A good pattern will have been evolved out of the experience (both successes and failures) and observations of a number of designers. It is susceptible to further refinement in a way that seeks to approach the underlying invariant, rather than simply recording more cases.

Additionally, the use of a pattern *language* holds out the possibility of involving the occupants of co-operative buildings, or the users of software interfaces in the design and modification of these artefacts. That we have not been able to address this possibility here does not indicate our lack of enthusiasm for it.

The **Interaction Feedback** pattern is shown to lead into another pattern:

### “Show Computer is Thinking” Pattern

**Situation:** operations can take a long time

**Problem:** people need to be warned when an operation is going to take a substantial amount of time, otherwise they may assume something has gone wrong. They are unwilling to sit and wait if the wait is to be fruitless, but on the other hand they do not want to confuse the system by reissuing commands unnecessarily.

**A solution:** give special feedback for lengthy operations. Examples would include changing the cursor to a watch or egg timer, showing a timeline, filling a time-bar and so on.

This could lead into the next pattern.

### “Just another 30 seconds” Pattern

**Situation:** operations can take a *very* long time.

**Problem:** sometimes people need to know what proportion of a task has been carried out, otherwise they may stop the operation just when it is about to end, or they may wait around by the screen unnecessarily when they might as well have gone for a cup of tea.

---

<sup>3</sup> For instance, “Do not use abstract or humorous designs for icons.” Carlow 2.1.4

**A solution:** use a measuring bar or percentage or text representation to tell user how much more work there is to do. Examples would include the mechanism on many Web browsers showing that, say, 45% of a site has been downloaded; alternatively, the “copy” operation in Apple Mac desktop tells the user how many files are left to be copied both in textual and graphical format.

## 3.2 Patterns in functional design

As in interface design, the design of the *functionality* of a piece of software lends itself to description in terms of patterns. This should not be a surprise. Alexander's patterns are intended to create buildings which allow users to live as they want: pattern-like thinking for software functionality design should similarly aim to build systems which incorporate just those functions which help a user to do what s/he wants.

### 3.2.1 Some Patterns

At its simplest, a functional pattern would be little more than an element of a requirements specification document together with a rationale from systems analysis or a workplace study, as in the first two examples.

#### “Provide Mail Merge” Pattern

**Situation:** people producing large numbers of letters on a word processor

**Problem:** people want to send out personalised letters to lots of different recipients without typing in the individual names

**A Solution:** provide a mail merge facility, allowing personal details from a database to be integrated into a standard letter.

#### “Provide Multiple Filing Systems” Pattern

**Situation:** in paper systems, people often have documents that need to be referenced in two places or under two or more headings. In real life offices people work round this either by creating a copy of the document and filing one under each heading, or putting a referring note in one of the locations.

**Problem:** this is a problem in the paper world as it involves long hours by the photocopier or messy bits of paper which have a tendency to get lost.

**A solution:** the same functionality can be provided in a lightweight way via software by using a "duplicate" or better a "create a link" command in an operating system.

#### “Think Twice” Pattern

**Situation:** the user may accidentally carry out an action which has serious consequences, such as deleting a file, erasing a disc and so on.

**Problem:** how to avoid disastrous errors while maintain fluidity in the interface.

**A solution:** if the consequences of a command can be grave and/or are irreversible, ask the user to confirm the command, but make "continue" or "OK" the default option.

A follow-on pattern might suggest that in the case of extremely risky operations, the default option at this point should be the more cautious one.

**Examples:** in the Macintosh interface, the "Empty Wastebasket" command displays a dialogue box containing the number of items in the waste bin, together with their size and gives a choice of "cancel" or "OK" with OK, the *less* cautious option as the default. However, the "Erase disk" command brings up a dialogue box containing the name, location and type of disc and gives a choice of "cancel" or "erase," with "cancel", the *more* cautious option, as the default.

### 3.3 Relevance in ergonomic design

When we move to the level of the physical artefact in which the software is embedded, the pattern language structure can be used to point out general contextualised problems and general solutions which can be implemented to fit specific situations.

#### 3.3.1 Some Patterns in Ergonomic Design

##### "Make a Computer that looks like a Filofax" pattern

**Situation:** business people want to carry their computers around with them.

**Problem:** business people tend to be relatively mobile, so heavy weights are impractical. They also tend to carry briefcases and paper-based objects such as reports, diaries and personal organisers.

**A Solution:** create a casing for the computer which lightweight, as handy as a Filofax and which business people can carry in the briefcase which is part of their "uniform".

##### "Make a computer like a pet mouse" pattern

**Situation:** a toy firm has developed an idea for a computer-based pet.

**Problem:** an electronic toy for children, like the portable computer, also needs to be portable. However, children tend not to carry briefcases (outside school, that is). Some have pockets, others not.

**A Solution:** build them something palm-sized and highly portable, either in a pocket, like a pet mouse, or on a cord around the neck

##### "Make a casing a gorilla can't destroy" pattern

**Situation:** a research project in animal understanding, which studied the abilities of a specific gorilla, Koko by name, in manipulating the American Sign Language.

**Problem:** the problem was that the computer had to be a Mac with touch-screen for functional reasons, but had to be able to withstand the attentions of the gorilla which included flinging it against a wall and inserting bananas through its vents.

**Solution:** a special housing in tough, gorilla-proof metal, described in more detail in Clark *et al.* (1990). This is perhaps an example of a pattern of the future. At the moment a solution has been found to a specific problem, but it is not hard to imagine how both problem and solution could be generalised. Casings for other difficult situations might use the same ideas in this pattern.

### 3.4 Patterns in the built environment

At yet another level we encounter problems or contexts in the physical world, which need a physical world solution but one which has a software element to it. Here we could imagine experimenting with Alexander's patterns unchanged as they should apply directly. In particular Alexander's patterns 146 - 252 on work situations, covering flexible office space, communal eating arrangements, small work groups, reception areas, waiting areas, small meeting rooms and half-private offices will have direct applicability to cooperative buildings in enabling people to meet and also to withdraw into privacy as they wish.

#### 3.4.1 *Some patterns*

Some patterns can be discerned at the level of furniture and consoles.

#### **“Attach a Document Holder” Pattern**

**Situation:** the user needs to be able to refer to a document while using the screen.

**Problem:** there often isn't enough room on the desktop (which is taken up by the computer) and anyway looking down at the document on the desktop would mean constantly shifting one's focus of attention with all the attendant disorientation problems that this implies.

**Solution:** attach a document holder to the side of the terminal

Others treat the wider workspace:

#### **“Sociable Terminals” Pattern**

**Situation:** a current workplace design, e.g. in a control room of some sort, allows co-workers to consult each other's work. For instance, colleagues glance at each other's screens when walking to the coffee machine and often pick up problems as they do so.

**Problem:** there are moves to introduce a system of small cubicles for workers. This would mean that the extra check afforded by glancing at a colleague's screen was no longer available, possibly lowering safety standards.

**Solution:** design the new layout perhaps as a horseshoe, making sure screens are big and readable and of common design.

This pattern might be linked to the following pattern, which refers to the need for co-workers to be in auditory rather than visual contact.

### "Let people overhear" pattern

**Situation:** a current workplace design, e.g. in a control room of some sort, allows co-workers to overhear each other's interactions, including telephone conversations, and sometimes to act on what they overhear. Sometimes this is trivial: the person on the telephone says "I'll just find a pen..." and person B hands one to them. At other times a rapid response is vital, as when person B picks up from person A's urgent tone on the telephone that an accident has occurred. This often means a quicker response to critical incidents than if the person on the telephone had to explicitly relay the information to his/her colleague.

**Problem:** as in the preceding pattern, there are moves to introduce a system of small cubicles for workers. This would mean that this benign overhearing was no longer possible.

**Solution:** Rethink the design and don't put people in boxes or out of eye contact or in completely soundproof headphones.

The current trend to carry out quasi-ethnographic studies of workplaces such as control rooms as a prelude to the redesign of collaborative software has led to the problem of finding a language in which the social scientists carrying out the study can structure their results for clients and designers. As Erickson (1997) has suggested, patterns may provide just such a form, as these examples briefly demonstrate.

## 4 Concluding Questions and Exhortation

Integrating systems into environments involves considering the links between many disparate levels of analysis: the relation between what the user wants to do and the functionality of the system, the design of the screen within the input/output device, the physical relation or interface between user and input/output device, and the placing of the device and the user(s) in the wider physical context. While these have traditionally been treated separately, Alexander's approach suggests that integration is not only possible but necessary.

Design Patterns give us a set of terms, concepts and values which have resonances at every level of the design of both software, hardware and the built environment. We have hardly scratched the surface in this paper, but we feel the approach is promising and that it would be foolish not to explore further. At the moment we find ourselves in the position of simply feeling that design patterns *should* be of use to designers. Their analytical and descriptive power is impressive, but what we really need are aids to *synthesis* not to *analysis*. One way to find out whether they are

really useful is, as Lea (1997) suggests, for people to stop writing *about* design patterns and to get down to the business of identifying and describing them in a public forum so that they can be smoothed and polished by the comments of peers and can eventually form part of the discourse of co-operative building design.

## References

1. Alexander, C., Ishikawa, S. & Silverstein, M. (1977). *A Pattern language*. Oxford, Oxford University Press.
2. Alexander, C. (1979). *The Timeless Way of Building*. Oxford, Oxford University Press.
3. Apple Computer, Inc. (1992). *Macintosh Human Interface Guidelines*, Addison-Wesley Publishing Co.
4. Carlow International, Inc. (1992). *Human-Computer Interface Guidelines*, NASA, Goddard Space Flight Centre, [http://groucho.gsfc.nasa.gov:80/Code\\_520/Code\\_522/ Documents/HCI\\_Guidelines/](http://groucho.gsfc.nasa.gov:80/Code_520/Code_522/Documents/HCI_Guidelines/) viewed Dec 18 1997.
5. Clark, M., Ferrara, T., Jones, D., Marion, A., Rose, K. & Yaeger, L. (1990) Koko's Mac 11. In: Laurel, B. (Ed.), *The Art of Human Computer Interface Design*. Addison Wesley. 95-102.
6. Cooper, A. (1995). *About Face: The Essentials of User Interface Design*. IDG Books.
7. Cross, N. (1984). *Developments in Design Methodology*. Chichester, John Wiley.
8. Cross, N., Christiaans, H., & Dorst, K., (Ed.) (1996). *Analysing Design Activity*. Chichester, John Wiley.
9. Erickson, T. *Report on Design Patterns Workshop CHI '97*. [http://www.pliant.org/personal/Tom\\_Erickson/](http://www.pliant.org/personal/Tom_Erickson/), viewed Nov 1,1997.
10. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). *Design Patterns*. Addison-Wesley.
11. Jones, J. C. (1970). *Design Methods*. New York, Wiley.
12. Kay, A. (1990). User Interface: A Personal View. In: Laurel, B. (Ed.), *The Art of Human Computer Interface Design*, Addison Wesley. 191-207.
13. Lakoff, G. & Johnson, M. (1980). *Metaphors We Live By*. Chicago University Press.
14. Lawson, B. (1980). *How Designers Think*. London, Architectural Press.
15. Lea, D. (1997). *Christopher Alexander: An Introduction for Object-Oriented Designers*. <http://g.oswego.edu/dl/ca/ca/ca.html>, viewed Nov 6, 1997
16. Microsoft Corporation (1987). *The Windows Interface: An Application Design Guide*, Microsoft Corporation
17. Tognazzini, B. (1993). Principles, Techniques and Ethics of Stage Magic and their Application to Human Interface Design. In: *Proceedings of CHI '93*. 335-362.