Interaction Design Patterns: Twelve Theses

Jan O. Borchers
Department of Computer Science
University of Linz, Austria
+49 731 502 4192
http://www.tk.uni-linz.ac.at/~jan/

ABSTRACT

This position paper was written for the CHI2000 Patterns Workshop, The Hague, 2–3 April, 2000. It is a revised and extended version of a paper for a patterns workshop of the British HCI Group in March 2000. It offers twelve statements outlining my position about patterns in human-computer interaction (HCI). The first, under "Roots", suggest how HCI, unlike software engineering, can adapt the original patterns idea from architecture. The second set, "Adaptation", shows how this concept can be expanded to cope with the dynamics and requirements of user interfaces. The "Users" section explains how HCI patterns lead to participatory design. The closing claims introduce some current work on using patterns in software engineering, HCI, and the application domain to enhance communication in interdisciplinary design teams, and outline how those pattern languages fit into the usability engineering lifecycle.

ROOTS

Claim 1 *Architecture is closer to HCI than to software engineering.*

The original intention of patterns as introduced by architect Christopher Alexander [1, 3] was to capture the essence of successful solutions to recurring design problems in urban architecture. According to Alexander, "a building or a town is given its character, essentially, by those events which keep on happening there most often" [1, p. 66], and the social patterns of activity in an environment determine the geometrical and structural patterns of that environment.

Consequently, Alexander's patterns of urban architecture describe aspects of the physical environment in which people live and work. The architect is the "designer", and the inhabitants are the "users" of these environments. *The artefact that the architect designs is something that its inhabitants directly interact with and live in.*

The task of HCI design is very similar to this: User interface designers shape a virtual environment in which the user works, or, in the case of 3-D virtual environments, may even live. As we know, users generally identify a software system with what its user interface shows them, without caring about its inner works. *The artefact that the user interface designer creates is something that its users directly interact with or even live in.*

In striking contrast to these two professions is the work of the developer, or software engineer: There is no doubt that she creates a system that has its own structure and architecture, but the only people who will ever "live inside" this architecture are software engineers! As with any technical system, it is not the goal of a software system to make its entire inner workings transparent and understandable for the user. Instead, its user interface is designed to create a consistent and easy-to-use abstraction from this inner complexity. *The artefact that the software engineer designs is something that its users should not have to directly interact with, and certainly will never live in.*

The direct connection between design work and user experience, therefore, is something that makes HCI and architecture have much more in common than architecture and software engineering. As Barfield et al. put it in their Interaction Design curriculum description:

> "Like the architect who views a building as an environment that defines certain potential behaviours of those who go there, the interaction designer focuses on behavioural patterns – on the way these patterns can shape people's lives." [4, p. 70]

Claim 2 *The Gang of Four book contains no patterns.*

Another important aspect of patterns as introduced originally is that they are readable and understandable by professionals and non-professionals alike. In fact, Alexander's patterns address both audiences, for example with his illustrations: At its very beginning, each pattern contains a detailed, realistic example photograph to quickly "sensitize" readers to the subject that the pattern addresses, even if they are architectural laymen. After the solution has been described, however, most patterns are illustrated once more in the more abstract, concise form of a sketch, leaving out unimportant details, and concentrating on the essence of the solution. This is a format that especially professionals will prefer over a crowded photograph to quickly grasp the idea of a pattern.

Naturally, the patterns Alexander presents do not make a layman an expert architect, and Alexander admits this in his current theories that require a "master builder" as well as patterns. But they do describe the essential issues to consider when creating a room, house, or neighbourhood layout. Moreover, the writing style in "A Pattern Language" is such that non-architects can easily read and understand most of the patterns presented, and try to apply them themselves. In fact, Alexander's goal was to give the inhabitants a language to express their own design preferences and to have a word (again)

in the design of their environments. But this approach is strikingly similar to the ideas of user-centred design as preached and occasionally even practised in HCI design work.

Contrast this with the most successful text in the software patterns community [12]: The "Gang of Four" book is without doubt a very valuable resource for aspiring programmers and software engineers, capturing successful solutions (and sometimes hacks) to recurring problems in object-oriented design.

But it is not readable by people from outside the software engineering profession. It does not describe aspects of an architecture that anybody else outside the profession can (or would like to) understand, let alone "live" in. In a way, software engineering has successfully adapted an important part of the pattern concept – creating a vocabulary of solutions to recurring design problems for communication within the community –, but has failed to carry over the more fundamental idea of supplying this language (and the power associated with it) to others involved in their projects, especially end users. And in fact, it may not even be interesting for those users to learn this language; they will not want to deal with those inner structures of the software system. (As an example, the GoF book contains code fragments to illustrate its concepts, which are unreadable for non-programmers, and which would probably translate to Alexander using static equilibrium calculations to illustrate some of his patterns.)

Moreover, as Jim Coplien pointed out at ChiliPLoP'99 [8], the book does not really describe a structured approach to learn the essential concepts of object oriented software design. The patterns are more like a collection of tricks than a structured treatment of the subject. Nevertheless, some object-oriented software design courses are taught simply by going through the GoF book!

It is interesting to note that recently, the Gang of Four were even "tried" for their book in a mockup process at the OOPSLA'99 conference. Jenifer Tidwell, author of the most ambitious HCI pattern language effort to date [21], has written an interesting comment on this process [22].

Claim 3 *HCI needs to derive its own pattern notion from the original sources.*

From the last two claims, it becomes clear that it will be more fruitful for HCI to take a fresh look at the original sources of the pattern idea, than to blindly adopt the software engineering notion of design patterns just because it appears to be a closer profession (which, as outlined above, it may not even be). The ideas included in the original pattern concept – user participation, and an understandable language for non-professionals – are too valuable and too close to the interests of HCI to be ignored.

It is important to see that the software engineering community has not ignored these aspects entirely. Many pattern experts have tried to carry Alexander's larger goals over to software engineering. But at OOPSLA'96, Alexander made it clear in his keynote speech that the majority of software patterns folk had overlooked something he considered fundamental in his work, something that he called "the Quality Without A Name", and that for user interfaces is best described by adjectives such as "intuitive", or "transparent":

> "Now, my understanding of what you are doing with patterns... It is a kind of a neat format, and that is fine. The pattern language that we began did have other features, and I don't know whether those have translated into your discipline. I mean, there was at root behind the whole thing a continuous mode of preoccupation with under what circumstances is the environment good. In our field that means something." [2]

The CHI'97 workshop [5] arrived at a similar result, agreeing that the *values* of a design school, its overall beliefs in what constitutes a "good" solution to a design problem, are very strongly represented inside a pattern language written by an author of that school.

Interestingly, while software engineering has picked up the pattern idea far more intensely than HCI, several classic HCI texts [19, 18, 11] already cite Alexander's pattern work as an influential source, while the first appearance of patterns in the object-oriented community dates back to OOPSLA'87 [6].

Claim 4 *HCI pattern languages are not Style Guides. HCI pattern languages are not Golden Rules. HCI pattern languages are not standards. But HCI pattern languages are about HCI.*

There are many formats that have been considered in the HCI community to effectively communicate design experience, usually to others from the same profession.

Style guides, such as [11], are a very efficient medium to ensure consistency within a single user interface environment, or "toolkit". They are concrete, and constructive, but do not carry over well to a different toolkit.

General guidelines, such as the Eight Golden Rules of Interface Design [20, p. 74], are useful to categorize a bad characteristic of an existing interface design, and to trace this back to the rule it breaks, but they miss concrete examples, and are not constructive (they do not help designers with options when they have to design a new system).

Standards, such as the ISO Multimedia User Interface Design Software Ergonomic Requirements [15], are a useful vehicle to establish rigid rules in a developing or mature field. But without references to concrete examples, and an informal writing style, they are not ideal to learn about design.

It is very useful to evolve these other forms of communicating design experience further. But when we talk about patterns, we should make sure that the essential characteristics,

the structure and components that define what a pattern is, are not forgotten. (Otherwise, why call them patterns?)

For example, each of Alexander's patterns describes a successful solution of a design problem. It contains name, ranking, picture, context with links to larger-scale patterns, problem statement and description, solution, diagram, and references to smaller-scale patterns, and it balances a set of opposing forces optimally in some way for the given context. HCI patterns should also contain those ingredients, or explain how and why they adapt this format for HCI (as I do below).

Along the same lines, it has to be clear what domains HCI patterns should address. Of course, all issues of interaction between people and machines that clearly fall into the area of HCI are to be addressed by patterns. They range from task representation, dialogue, navigation, information and status representation, down to layout, device aspects, physical interaction, etc.

However, issues such as social interaction should be treated with caution: While it is very important that these aspects are dealt with when designing a user interface, patterns about them would not address the human-computer dialogue directly. The pattern format is without doubt extremely suitable to model aspects of human-human interaction as well as human-computer interaction. However, to keep the notion of HCI patterns focused, it may be necessary to deal with those "HHI patterns" under a separate name. (Otherwise, why call our patterns *HCI* patterns?)

Actually, this is a discussion about what fields belong to HCI. At the Interaction Patterns Hot Topic at ChiliPLoP'99 [8], the same problem arose, because to define what Interaction Design Patterns should describe, we needed to define what Interaction Design as a discipline contains. The report contains an initial attempt at a definition:

> An Interaction Pattern Language generates space/time interaction designs that create a system image close to the user's mental model of the task at hand, to make the human-computer interface as transparent as possible. [8]

At the INTERACT'99 workshop, we started with this definition, but arrived at a more "user-centred" form:

> The goals of an HCI pattern language are to share successful HCI design solutions among our colleagues, and to provide a common language for HCI design to anyone involved in the design, development, evaluation or use of interactive systems. [14]

It would be interesting, and helpful for people from outside the workshop, to merge and expand these two attempts into a more detailed definition.

ADAPTATION

Now that I have outlined what I believe should be the starting point to define what HCI patterns are, I will discuss how I think the original pattern concept needs to be changed and extended to become adequate for capturing user interface design experience.

Claim 5 *Architectural pattern languages need to be extended by the notion of temporal structure to deal with HCI.*

A pattern language is much more than the sum of its individual patterns. The links from patterns addressing large-scale design issues down to small-scale design details help the reader and prospective designer to find the next important pattern as she refines her design. In architecture, the resulting hierarchy is quite simply ordered by geometrical size: Patterns dealing with the general layout of an entire neighbourhood are higher up in the hierarchy than patterns dealing with the question of how to split up an individual house into rooms.

This simple organizing principle ignores one major dimension: time. This works reasonably well for architecture, as the artefacts created (buildings, streets, etc.) do not change themselves substantially over time. Only the events taking place within those environments change over time, and Alexander uses such sequences of events (such as traffic intensity over the course of a day) as a single aspect that influences the design of an environment at the geometric level the pattern addresses.

This approach does not work for HCI, because the artefacts we create do change substantially over time, follwing the tasks they support. To give a simple example: A railway information kiosk changes from a start page, to a page giving train type options and travel time input fields, to another page displaying possible train connections, etc. In other words, we design user interfaces along a time dimension as well as along two (or three) spatial dimensions.

Therefore, the ordering principle of spatial size has to be expanded into an ordering principle of spatial and temporal expansion. One obvious solution is to put time at the top of the hierarchy, according to the large-scale notion of tasks: First, the designer thinks about what the complete task looks like, what objects and procedures it contains, and how it can be supported by a series of interactions, or dialogues. Then she goes into detail for each of those steps, designing those shorter dialogues, until each step in the dialogue sequence (or rather graph) is designed with its interaction objects, layout, and spatial geometry.

Other ordering principles are possible, for example those that are more oriented toward the design process itself; we discussed those possibilities in some detail at the INTERACT'99 workshop [14], but more work needs to be done in this area.

Claim 6 *The structure and components of individual patterns need to take the temporal dimension into account.*

The medium that architectural patterns use to sensitize the reader to the subject of a pattern is a photograph. The INTERACT workshop arrived at the conclusion that time-based media, such as animated scenes or video clips of an interaction, will be necessary to introduce the ideas of HCI patterns that deal with dynamic aspects of a dialogue. While an interactive mini-application showing a certain interface concept (such as Balloon Help) "in use" would also be possible as example, it was considered too complicated to understand for non-professionals (for example, a reader would have to discover somehow that balloons appear in the example when a pointer is moved over an object).

Similarly, the architectural sketches will need to be replaced by time representations such as storyboards for such HCI patterns. Video material would be less useful here, since it takes longer to grasp when a professional browses a pattern collection.

Claim 7 *HCI patterns need to give empirical evidence of their validity.*

A point that is often criticized in Alexander's work is the fact that he hardly gives any "hard" empirical data to support the superiority of his solutions. In some sense, this is understandable, since the ultimate measure he uses is whether people actually say that they *feel good* in an environment or not [1, p. 289–297]. Alexander argues that, while economical, practical, and stylish opinions usually diverge, the criterion of feeling good in an environment returns high levels of agreement between people for good, timeless solutions (for example, using brick walls vs. prefabricated modular wall panels).

Nevertheless, HCI is a discipline that has a strong foundation in empirical data from user tests and preferences, and these scientific methods need to be reflected in an HCI pattern language to make it valid and acceptable for the HCI community.

However, it has to be ensured that endless statistical tables do not make the pattern unreadable, especially for non-professionals.

Therefore, the *Examples* section of a pattern should include brief summaries of results from representative user studies, with references to the original publications. This information will satisfy the demands of scientific rigour of the professional and researcher, but also show the amateur that the suggested solutions have been tested empirically.

USERS

With these adaptations of the original pattern concept in place, I would like to point out some issues that I think are important characteristics of patterns for the end users of a system.

Claim 8 *HCI patterns must be readable by users.*

As explained above, one of the original goals of patterns was to supply the "users" of environments with a language to influence environmental design decisions.

If HCI aims at a similar user participation in the design process, HCI patterns must be written in a form understandable even by UI design laymen. Arcane notations, while useful for the expert, must be turned into readable prose. After all, patterns are ultimately a *literary form*, with strong rules for format, content, and style, all to maximize readability.

Claim 9 *HCI patterns will take power away from HCI designers and put it into the hands of the users.*

His colleagues did not like Alexander's ideas particularly when he first published them. One of the reasons for this is that those pattern languages aimed at empowering inhabitants to participate in the design process. This implied that architects would have had to give away some of their power, and incorporate the opinions of non-professionals. Consequently, few architects have used those patterns to give them to customers and talk with them about their design ideas. Instead, Alexander's pattern language book has gained a large following among people wishing to redecorate their house or garden, essentially working as "amateurs".

If HCI takes the patterns idea seriously, then its adoption must go hand in hand with a stronger participation of users in the user interface design process. It appears that there is a more welcoming atmosphere for such measures in HCI than there was in architecture, which can be seen in the advent of participatory and user-centred design methods in recent years. But ultimately, it may be well possible that not UI design professionals, but rather end users will pick up a copy of those HCI patterns and use them to redefine, customize and improve their own private computing environments.

INTERDISCIPLINARY DESIGN

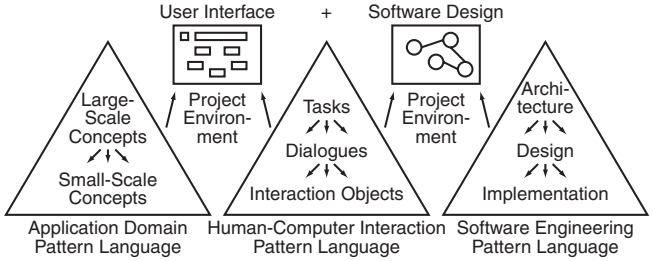
The final two claims lead to my recent work in HCI patterns, which extends the pattern concept to further domains of application and suggests a new participatory design method.

Claim 10 *Patterns can model many application domains.*

Patterns have been a successful tool to model design experience in architecture, in software design (with the limitations discussed here), and, as existing collections show, also in HCI [21]. Other domains have been addressed by patterns as well. Patterns have been used to describe business domains, processes, and tasks to aid early system definition and conceptual design [13]. Even composing or improvising music can be considered a design activity, and some experience be expressed in pattern form [10].

Indeed, there is no reason why the experience, methods, and values of any application domain cannot be expressed in pattern form, as long as activity in that application domain includes some form of design, creative, or problem-solving work.

Claim 11 *Using patterns in software architecture, interaction design and the application domain of a project can improve communication in interdisciplinary design teams.*



This is the essence of my current research: If patterns are written with their original qualities, including readability by other professions, in mind, then the groups who have to work together in a software design project could understand each other much better. Software engineering could express its methods, experience and values about software architecture in pattern form. Similarly, HCI designers could rewrite their style guides, standards, golden rules, etc. in the form of an HCI pattern language. And finally, users, or other experts from the application domain of a project, could use the same format to express their own knowledge about the application domain.

Claim 12 *The use of patterns in the various domains can be mapped to most phases of the usability engineering life cycle.*

Nielsen [17] suggests a *usability engineering life cycle* consisting of eleven phases. The following table shows which parts of the pattern languages can be used in which phase.

Phase	Usability Engineering Life Cycle	Pattern use
1.	Know the user (characteristics, current & desired tasks, functional analysis, user & job evolution)	Extract application domain experience as pattern language
2.	Competitive analysis (examining existing products)	Generalize good UI solutions into HCI patterns
3.	Setting usability goals (financial impact analysis, prioritizing design goals)	Use competing goals as forces in abstract HCI patterns
4.	Parallel design (several initial designs by independent teams)	Use general HCI design patterns (maybe from book) as common design guidelines for the teams
5.	Participatory design (actively involving users in the design process)	Application domain expert (user) and HCI designer exchange their pattern languages for better mutual understanding
6.	Coordinated design of the total interface (Consistency within and across products)	Lower-level HCI design patterns, including project-relevant, concrete examples, communicate the common look and feel efficiently
7.	Apply guidelines and heuristic analysis (style guides, standards, and guidelines)	Patterns improve upon those formats because of their standard format, hierarchical networking, inclusion of examples, and discussion of problem context as well as solution
8.	Prototyping	Software design patterns express the standards, components, and specific project ideas of the development team in a way better understandable by the HCI experts
9.	Empirical testing (user tests)	Problems discovered can be related to applicable patterns to solve the problem (vocabulary function)
10.	Iterative design (improve prototypes, capture design rationale)	HCI and software patterns (constructive, unlike guidelines) inform designers about design options at each point, and help capturing the space of design options explored (structural design rationale) – possibly with anti-patterns for bad solutions.
11.	Collect feedback from field use	Use application domain pattern language again as common vocabulary between UI experts and users. Use feedback to strengthen successful HCI and software patterns, and to re-evaluate suboptimal ones.

The basic pattern idea is not too hard to convey, and the patterns will evolve over time, becoming more stable with every new project. But the important thing is that, once the structure of expressing concepts is the same for all participants, they can exchange their languages, and use them all together as a *lingua franca* within the team. In discussions, important concepts can be recalled with a single word (the name of the pattern), and there is a documented base of methods and values that everybody can refer to. This may help to bridge the interdisciplinary gaps that often still prevent successful cooperation within such teams [16]. These ideas, and our application to some of our own projects, are discussed in more detail in [7] and [9].

I hope that these twelve theses can help to create a lively discussion about the state of HCI patterns at the workshop. Additional information is available from the *HCI Patterns Home Page* at ⟨http://www.tk.uni-linz.ac.at/~jan/patterns/⟩.

ABOUT THE AUTHOR

Jan Borchers is a researcher at the Telecooperation Research Group, University of Linz, Austria, and works as visiting scientist and lecturer at the University of Ulm, Germany, teaching courses in Designing Interactive Systems, Web Design, and Telecooperation. He has carried out project management and user interface design on a number of projects dealing with interactive exhibits, including an award-winning music exhibit, and works on a pattern-based approach to interdisciplinary interaction design. He has published his results at CHI, IEEE ICMCS, and other international HCI conferences, and in IEEE Multimedia, Computers & Graphics, the SIGCHI Bulletin, and other journals. He participated in the HCI patterns workshop at ChiliPLoP'99, and co-organized those at INTERACT'99 and CHI 2000.

References

[1] Christopher Alexander. *The Timeless Way of Building*. Oxford Univ. Press, 1979.

[2] Christopher Alexander. OOPSLA'96 keynote speech, 1996. Conference video.

[3] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. Oxford University Press, 1977.

[4] Lon Barfield, Willie van Burgsteden, Ruud Lanfermeijer, Bert Mulder, Jurriënne Ossewold, Dick Rijken, and Philippe Wegner. Interaction design at the Utrecht School of the Arts. *SIGCHI Bulletin*, 26(3):49–79, 1994.

[5] Elisabeth Bayle, Rachel Bellamy, George Casaday, Thomas Erickson, Sally Fincher, Beki Grinter, Ben Gross, Diane Lehder, Hans Marmolin, Brian Moore, Colin Potts, Grant Skousen, and John Thomas. Putting it all together: Towards a pattern language for interaction design. *SIGCHI Bulletin*, 30(1):17–23, January 1998.

[6] Kent Beck and Ward Cunningham. Using pattern languages for object-oriented programs. Technical Report CR-87-43, Tektronix, Inc., September 17, 1987. Presented at the OOPSLA'87 workshop on Specification and Design for Object-Oriented Programming.

[7] Jan Borchers. Designing interactive music systems: A pattern approach. In *Proceedings of the HCII'99 8th International Conference on Human-Computer Interaction*, Munich, Germany, August 22–27, 1999.

[8] Jan Borchers. CHI meets PLoP: An interaction patterns workshop. *SIGCHI Bulletin*, 32(1), January 2000. Workshop at ChiliPLoP'99 Conference on Pattern Languages of Programming, March 16–19, 1999, Wickenburg, AZ.

[9] Jan Borchers. A pattern approach to interaction design. Paper submitted to DIS 2000, New York, August 17–19, 2000.

[10] Jan Borchers and Max Mühlhäuser. Design patterns for interactive musical systems. *IEEE Multimedia*, 5(3):36–46, 1998.

[11] Apple Computer. *Macintosh Human Interface Guidelines*. Addison-Wesley, 1992.

[12] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[13] Åsa Granlund and Daniel Lafrenière. PSA: A pattern-supported approach to the user interface design process. Position paper for the UPA'99 Usability Professionals' Association Conf. (Scottsdale, AZ, June 29–July 2, 1999), 1999.

[14] Richard Griffiths, Lyn Pemberton, and Jan Borchers. Usability pattern language: Creating a community. Workshop at INTERACT'99 (Edinburgh, Scotland, August 30–31,1999); report in preparation, 2000.

[15] ISO 14915. Multimedia user interface design software ergonomic requirements, 1998.

[16] Scott Kim. Interdisciplinary cooperation. In Brenda Laurel, editor, *The Art of Human-Computer Interface Design*, pages 31–44. Addison-Wesley, 1990.

[17] Jakob Nielsen. *Usability Engineering*. Morgan Kaufmann, San Francisco, 1993.

[18] Donald A. Norman. *The Psychology of Everyday Things*. Basic Books, NY, 1988.

[19] Donald A. Norman and Stephen W. Draper. *User-Centered System Design: New Perspectives on Human-Computer Interaction*. Lawrence Erlbaum Associates, Hillsdale, NJ, 1986.

[20] Ben Shneiderman. *Designing the User Interface (3^{rd} Ed.)*. Add.-Wesley, 1998.

[21] Jenifer Tidwell. Interaction design patterns. PLoP'98 Conference on Pattern Languages of Programming, Illinois, extended version at http://www.mit.edu/~jtidwell/interaction_patterns.html, 1998.

[22] Jenifer Tidwell. The Gang of Four Are Guilty. http://-www.mit.edu/~jtidwell/gof_are_guilty.html, 1999.