PUX: Patterns of User Experience

Alan F. Blackwell
University of Cambridge Computer Laboratory

Sally Fincher
University of Kent at Canterbury

Manuscript to appear in *Interactions* magazine, Mar/Apr 2010.

Alexander's pattern language provided a way of raising the level of discourse about buildings from a concrete to a new abstract level of description [1]. Rises in abstraction level happen regularly in all fields, but the key difference in Alexander's work was that his abstract descriptions were founded in user experience, not in abstract descriptions of building construction (engineering) or of ornament (style). This is in contrast to recent developments in software patterns, as noted by Molly Steenson in her recent Interactions article[2].

In talking about software, it is easy to get confused about the distinctions between abstract and concrete, because so much about software seems abstract. As a result, the adaptation of pattern languages to software has lost the key contribution of Alexander's work, which was to throw attention onto the users. Software patterns, despite being inspired by Alexander's work, emphasise abstract descriptions of construction and of ornament, not abstract descriptions of user experience. It's time to change that. This article tracks down the history of where we took a wrong turning, and proposes an alternative way forward.

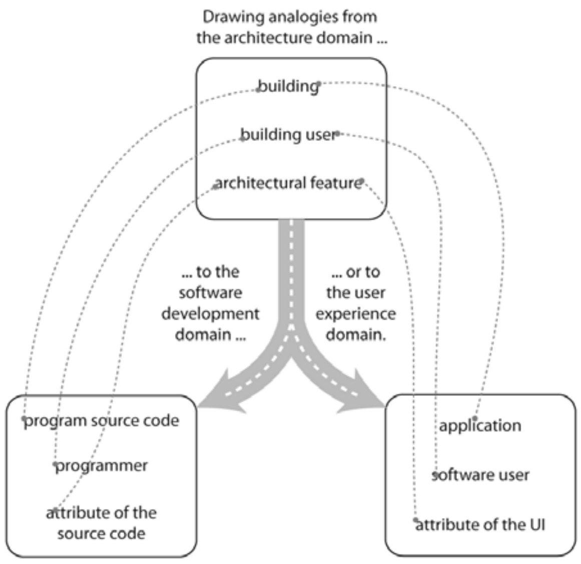
"Traditional" software patterns are concerned with user experience, but mostly with the user experience of *programmers*. That perspective may sound strange in an HCI context, but it helps explain the popularity of pattern languages in the programming community. If we move from the object world of technical software features to the human experience of structured information – we refocus attention on ways of working, not widgets. Our aim is a pattern language in the sense intended by Christopher Alexander, but a pattern language of user *experience* design rather than a pattern language of user *interface* design. This lets us escape shallow understanding of user experience in terms of affect and passive consumption (architects describe this as ornament [3]), to the ways that users perceive and build information structures.

Before the publication of the 'Gang of Four' book that popularised software patterns [4], Richard Gabriel described Christopher Alexander's patterns in 1993 as a basis for reusable object-oriented software in the following way:

> *Habitability is the characteristic of source code that enables programmers, coders, bug-fixers, and people coming to the code later in its life to understand its construction and intentions and to change it comfortably and confidently.*

It should be clear that, in our context, a "user" is a programmer who is called upon to maintain or modify software; a user is not (necessarily) the person who uses the software. In Alexander's terminology, a user is an inhabitant [5]

Gabriel offered an explicit analogy between architecture and software, the nature of his analogy is shown in figure 1.


Drawing analogies from the architecture domain ...
building
building user
architectural feature
... to the software development domain ...
... or to the user experience domain.
program source code
programmer
attribute of the source code
application
software user
attribute of the UI

In 1993 the time was ripe, after a sustained period of great enthusiasm for HCI and usability, for addressing the usability needs of programmers. It is little wonder that pattern languages of programming became so popular, as evidenced by sales of the Gang of Four book, and the many PLoPs (Pattern Languages of Programming) meetings.

However, in design terms these patterns exist in an 'object world'[6], primarily shared by engineers. The Gang of Four book included many nice engineering solutions, including elegant tricks for building user interfaces. But Alexander's ideas had not been primarily about solving technical problems – his concern was with the experience of the building users, not the engineers.

Alexander proposes homes and offices be designed and built by their eventual occupants. These people, he reasons, know best their requirements for a particular structure. We agree, and make the same argument for computer programs. Computer users should write their own programs. Kent Beck & Ward Cunningham, 1987 [7]

Beck and Cunningham originally proposed the creation of pattern languages for software development, not to support software engineers but end-user programmers. They were concerned with Smalltalk, a language originally conceived by Alan Kay as an end-user programming environment that would support new kinds of artistic and creative experience for users including children and other non-professional programmers. The Smalltalk programming environment itself included many UI innovations, which later evolved into successful interfaces of the Xerox Star, the Macintosh and Windows [8]. But why did end-user programming not become as popular as the interactions that were designed to support it?

We took a wrong turn

It is our contention that there was a collective confusion between ground and field: that the cool features of the Xerox Smalltalk environment – windows, icons, mice – appeared to engineers as though these widgets *were* the invention, irrespective of their application to create a transformed user experience. Subsquently, user interface 'patterns' have continued to capture ways that engineers compose these (and other) widgets to build functional UIs. But this is missing the point. Using the Smalltalk widgets in that way has not given users the power to conceive and restructure their own experiences, and many GUI systems became even less flexible than the command line interfaces that preceded them.

The application of interaction patterns has remained stuck in the 'object world' – they are still concerned with the successful design and *engineering* of a user interface. User interface patterns that emphasise technical solutions are extremely valuable, but they are not the kind of design pattern that Alexander envisaged; they are not primarily concerned with the user's experience of the designed product, and neither do they empower end-user programming in the way that Beck and Cunningham hoped to achieve.

We propose that. rather than describing users' encounter with specific technical features, it is possible to use patterns to describe user experience with the whole class of structured information systems. This ought to encompass users both as consumers of standardised interface designs, and as empowered to customise and modify the structure of information. Spreadsheets, content management systems, word processor macros and home network configuration are tools that empower end-users with the capability of programmers, giving them user experiences that have the fundamental characteristics of programming[9]. The 'patterns' here are not a specific way of building a UI, but a language for describing user experiences with structured information.

What would a user experience pattern look like?

One classic example of such a pattern in user experience was noted 20 years ago by Thomas Green. He had found 'a sticky problem for HCI'[10], in that neither theoretical accounts of user behaviour, nor designer's expectation of how users ought to behave, allowed for the fact that users might want to change their mind. Many systems were designed with the assumption that the user would have a coherent and complete plan, and would be able to follow that plan when creating an information structure (for example a travel itinerary, a lesson plan, or a database schema).

As a result of following that mistaken assumption, designers built user data-management tools in which it was relatively easy to transcribe a pre-formed plan into the information system, but rather difficult to change the structure of that plan after it had been entered. For example, it is often the case that dependencies within an information structure mean that changing one thing requires change in another, which in turn requires another, and user empowerment falls like a sequence of dominos in a series of 'knock-on' changes. For Green, this was the sticky problem – such systems were 'viscous', so that the user experience of making changes to your plan felt like wading through treacle.

It was ironic that software designers were in the habit of making viscous systems, despite the fact that programmers themselves appreciate tools that allow them to readily explore alternatives, and to change aspects of the structure fluently and flexibly, for example tools for type inference, incremental compilation or refactoring. Green suggested that the same desirable flexibility in changing information structures should be offered to all users, and that designers should be alert to the possibility that they might be imposing high viscosity on users that they would not accept for themselves.

Viscosity does not describe a single technical feature of the user interface, but rather a user experience that spans multiple design decisions. Green presented that experience in negative terms as a problem for users, but it could be described in more Alexandrian terms as a template for positive experience: 'You can change your mind'. It has sub-species, some more familiar than others: 'You can change your mind immediately' is a user experience pattern that corresponds to the feature-oriented description 'undo'; while 'You can change your mind about the structure you are making' alerts designers to the kind of viscosity that has proven such a compelling example of the Cognitive Dimensions of Notations framework [11,12,13].

User experiences in representational systems

Alongside the experience of changing your mind, users of information structures soon become familiar with patterns such as 'See how elements depend on each other', which Green and Petre advocated when describing the dangers of *Hidden Dependencies*. Support for desirable user experiences most often presents designers with trade-offs. For example, recording and visualising dependencies tends to increase viscosity, because the explicit links and relationships make changes more laborious (as in visual programming languages, where the dependencies are laid out as lines between components, but the proliferation of lines makes it hard to move components around).

A further example based on Petre's work is the pattern of 'Freedom to leave informal notes' (which Green and Petre called *Secondary Notation*). Users often want to express or record things that the designer hasn't anticipated. Freedom to make comments, add reminders, make decorations, record vocabulary changes, or format choices that have no semantic interpretation within the system can be adapted by users for many purposes. A simple but useful case is the 'Notes' box when requesting a print copy of a map from Google Maps. It's often useful to add personal notes, independent of formal address and navigation instructions. If the system insisted on interpreting and formatting these, then it wouldn't be so useful.

Today we can see many collections of such "patterns", most inspired by Green's insights. Thsee have included patterns of user experience for collaborative meetings over an information structure[14], for information structures using tangible representations[15], for programming APIs[16] and for the kind of visual language created for end-user programming[12]. Many are derived from user accounts, and immediately recognisable to users, such as the experience of getting a gestalt view of the whole structure, or of making ambiguous marks that can help you see the problem differently[17]. However, none of these extensions of the Cognitive Dimensions framework has been presented to designers in the way we propose, as a pattern language, although one of us noted the potential parallels several years ago[18].

Patterns for architects; patterns for builders

Architects and builders have separate concerns from each other – the architect is ultimately solving a human problem, and the builder a technical problem (although both draw expertise across the boundary). The primary concern of a builder is in the 'object world' of building construction, whereas a successful architect is focused on empathy with the users of the building. In the same way our proposed patterns of software user experience are intended as resources for experience designers, but this means that they may not be seen as directly useful to interface developers, in the way that more concrete interaction patterns would be.

As a rule of thumb, anyone who regularly refers to pattern languages of programming is not likely to be the intended audience for patterns of user experience. Patterns of user experience are, however, more closely related to the architectural interpretation of Alexander's work. The 'internal' design patterns so popular in the software patterns community might be compared to a particular pattern of screws and brackets with which two beams can be securely connected, or a particular arrangement of fuse, switches and sockets by which occupants of a house can safely interact with a bathroom lighting circuit. The latter can be compared more directly to previously published HCI interaction patterns such *Action Button* and *Wizard* in the "basic interactions" section of van Welie's Interaction Design Pattern Library[19], which guide designers creating those atomic interactions: these are concerned with implementation detail rather than the descriptions of building experience so characteristic of Alexander's Timeless Way[20].

User experience as a Pattern Language

Philosophy of a pattern language

Christopher Alexander conceived his pattern language through a fundamental concern with user experience. In particular, he drew attention to aspects of user experience that extended beyond purely technical considerations, expressing regularities in user experience that were not obvious to other practitioners. Consider pattern 159 - *Light on Two Sides of Every Room* - which observes that people prefer rooms having natural light from two sources. As noted in [21] *Light on Two Sides of Every Room* is not "obvious," whereas *Build a Room with Windows* would have been obvious. And the technical 'object world' details of how *Light on Two Sides of Every Room* should be achieved by the builder – through use of materials, construction, integration into the wall – are not even described.

> *[Builders] can use this solution a million times over, without ever doing it the same way twice. (1, page x)*
>
> *Software patterns have evolved independently of their architectural origins, and even architects find this worthy of comment. A recent architectural publication notes '[Software] patterns are also independent of the software users' requirements and refer to categories that are more important to the software's programmer.' [22]*

Components of a pattern language

The Cognitive Dimensions of Notations framework has become a valuable tool for specialist applications, and especially for addressing the usability of programming tools, but this concern for the needs of programmers seems to have discouraged broader recognition of how universal these patterns in user experience really are. In particular, the ongoing search for a more formal basis ("What is the space in which these are dimensions?" "What is their cognitive base?", "How can the interaction patterns be formalised?") has distracted attention from the simple need to record and disseminate experience patterns.

Because these aspects of experience depend on information structures rather than simple visual features, and because they are experienced over the course of time rather than in direct reading or manipulation of a static display, it is not always easy to point to a specific piece of a user interface and say 'there it is'. Viscosity gained currency as a descriptive term because it resonated with people who shared (the frustration of) that specific experience. But many of the dimensions are not so readily recognised. This article marks a starting point for describing them in terms of structures in the user's experience, rather than as formal principles. For example, the relevant evidence could be presented as narratives, together with guidance helping the practitioner understand the ways that those narratives arise from, are supported by, or compensate for, features of the environment. This is work in progress and we welcome feedback toward our goals.

Patterns of User Experience can empower users

The key insight for thinking about abstract experience of representational systems is to recognise that the users of systems are ultimately concerned with navigating and configuring an information structure, just as users of a building are ultimately concerned with navigating and configuring the structure of space.

The essential benefit from pattern languages of user experience, for the HCI profession, should be to understand what kind of experiences people have with information structures. The patterns that we have described above – changing your mind, seeing dependencies, leaving informal notes – should be key concerns for designers of systems that offer users the power to configure and customise software for themselves.

The technical focus in the past on programming pattern languages has led to a focus on specific UI widgets and engineering concerns, such that the pattern language community has lost the perspective of empowering users to work with their own information structures. It is time to recover that focus, by collecting and disseminating patterns of user experience with structured information. We could apply such a pattern language to help us design humane systems, rather than being distracted by the changing technical structures and ornament that arrive with each generation of UI renderings.

(Footnote) references

[1] Alexander, C., Ishikawa, S. and Silverstein, M. A pattern language: towns, buildings, construction. Oxford University Press, New York (1977).

[2] Steenson, M.W. Problems before patterns: a different look at Christopher Alexander and pattern languages. Interactions 16(2) (2009) 20-23.

[3] Vrachliotis, G. (2009). And it was out of that that I began dreaming about patterns ...” On thinking in structures, designing with patterns, and the desire for beauty and meaning in architecture. In A. Gleiniger & G. Vrachliotis (eds). Pattern: Ornament, structure and behavior, pp. 25-39. Basel: Birkhäuser.

[4] Gamma, E. Helm, R. Johnson, R. and Vlissides, J. Design Patterns: Elements of reusable object-oriented software. Addison-Wesley. (1994).

[5] Gabriel, R.P. Habitability and piecemeal growth. Journal of Object-Oriented Programming (February 1993), pp. 9-14. Also published as Chapter 2 of Patterns of Software: Tales from the Software Community. Oxford University Press 1996. Available online http://www.dreamsongs.com/Files/PatternsOfSoftware.pdf

[6] Bucciarelli, L.L. (1996). Designing Engineers. MIT Press.

[7] Beck, K. and Cunningham, W. Using pattern languages for object-oriented programs. Tektronix, Inc. Technical Report No. CR-87-43 (September 17, 1987), presented at OOPSLA-87 workshop on Specification and Design for Object-Oriented Programming. Available online at http://c2.com/doc/oopsla87.html (accessed 17 September 2009)

[8] Blackwell, A.F. The reification of metaphor as a design tool. ACM Transactions on Computer-Human Interaction (TOCHI), 13(4) (2006), 490-530.

[9] Blackwell, A.F. First steps in programming: A rationale for Attention Investment models. In Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments(2002), pp. 2-10.

[10] Green, T.R.G. The cognitive dimension of viscosity: a sticky problem for HCI. In D. Diaper, D. Gilmore, G. Cockton and B. Shackel (Eds.) Human-Computer Interaction-INTERACT’ 90. Elsevier. (1990).

[11] Green, T. R. G. Cognitive Dimensions of Notations. In L. M. E. A. Sutcliffe (Ed.), People and Computers V. Cambridge: Cambridge University Press. (1989).

[12] Green, T. R. G., & Petre, M. Usability Analysis of Visual Programming Environments: a ‘cognitive dimensions’ framework. Journal of Visual Languages and Computing, 7 (1996), 131-174.

[13] Blackwell, A.F. and Green, T.R.G. Notational systems - the Cognitive Dimensions of Notations framework. In J.M. Carroll (Ed.) HCI Models, Theories and Frameworks: Toward a multidisciplinary science. San Francisco: Morgan Kaufmann (2003), 103-134.

[14] Bresciani, S., Blackwell, A.F. and Eppler, M. A Collaborative Dimensions Framework: Understanding the mediating role of conceptual visualizations in

collaborative knowledge work. Proc. 41st Hawaii International Conference on System Sciences (HICCS 08) (2008), pp. 180-189.

[15] Edge, D. and Blackwell, A.F. Correlates of the cognitive dimensions for tangible user interface. Journal of Visual Languages and Computing, 17(4) (2006), 366-394.

[16] Clarke, S. Measuring API usability. Dr. Dobb's Journal, Special Windows/.NET Supplement (May 2004).

[17] Blackwell, A.F., Britton, C., Cox, A. Green, T.R.G., Gurr, C.A., Kadoda, G.F., Kutar, M., Loomes, M., Nehaniv, C.L., Petre, M., Roast, C., Roes, C., Wong, A. and Young, R.M. Cognitive Dimensions of Notations: Design tools for cognitive technology. In M. Beynon, C.L. Nehaniv, and K. Dautenhahn (Eds.) Cognitive Technology 2001 (LNAI 2117). Springer-Verlag (2001), pp. 325-341

[18] Fincher, S. Patterns for HCI and Cognitive Dimensions: two halves of the same story? In Proc. 14th Workshop of the Psychology of Programming Interest Group (PPIG 14) (2002), pp.156-172.

[19] van Welie, M. Interaction Design Pattern Library: http://www.welie.com/patterns/index.php

[20] Alexander, C. (1978). The timeless way of building. Oxford University Press.

[21] Fincher, S. and Utting, I. Pedagogical patterns: their place in the genre. In Proc. 7th Ann. Conf. on Innovation and Technology in Computer Science Education (ITiCSE) (2002), pp. 199-202.

[22] Scheurer, F. Architectural algorithms and the renaissance of the design pattern. In A. Gleiniger & G. Vrachliotis (eds). Pattern: Ornament, structure and behavior. Basel: Birkhäuser (2009), pp. 41-55.