Teaching Usability Design Through Pattern Language

Richard N. Griffiths
University of Brighton
Brighton BN2 4GJ UK
+44 1273 642477
r.n.griffiths@brighton.ac.uk

Lyn Pemberton
University of Brighton
Brighton BN2 4GJ UK
+44 1273 642476
lp22@brighton.ac.uk

ABSTRACT

For interface designers to produce really usable software they require both knowledge of tools and methodologies and attitudinal, aesthetic and creative characteristics. Usability design patterns, based on the ideas of Christopher Alexander, offer an approach to this educational need. We discuss three approaches to incorporating design Patterns into teaching: teaching about Pattern language, discovering patterns and teaching through Pattern language. We conclude that developing an ability to see the world of usability design as patterns is more important than knowing a canonical set of particular patterns, but that this is an extremely difficult skill to acquire.

Keywords
Usability, patterns, pattern language, design, teaching.

INTRODUCTION

Challenges in teaching usability design

Designing usable software is difficult, and teaching others how to do it is worse! Although useful methodologies exist, it is not possible to teach someone simply “how to do it”. To be capable of doing more than producing Microsoft clones, student designers need a broader approach to the task, one that has important attitudinal, aesthetic and creative components.

Humility towards the end user is a particularly important attitude to cultivate, as slogans such as “The designer is not the user” and “Know your users” recognise. However, the interface designer may also need to challenge the user’s ideas with a broader view of good practice culled from the growing body of expertise. Being able to communicate with the user about software presents a challenge and requires careful thought about the choice of representation for the evolving design [4].

A sense of the whole quality of an interface design must be developed requiring the awareness of qualities such as, affordance, conviviality, fitness to task and appropriate stance. This sense is nascent in most experienced users of computer systems, but typically expressed negatively: “This interface is about as friendly as a cornered rat!” Like the palate of a wine connoisseur, it needs to be “educated”.

.

Creativity is also required of the interface designer, or else where are truly elegant solutions to come from? However, we must distinguish between a genuinely creative design, which elegantly solves a problem, and a solution which simply uses a novel technique or component. An ability to recognize the appropriate place for a previously identified solution, perhaps involving a refinement of a crucial detail to make it solve the new problem, is also a creative act. It exemplifies the routine creativity required of an expert designer and stems from a creative structuring of their deep knowledge of the domain. Assistance in developing this richly interconnected fund of knowledge is required. While guidelines are a gesture towards this craft knowledge, their explicit structuring is gross and often opaque.

Pattern Language

Pattern Language [1] is a way of representing and accumulating knowledge of good design. It was originally applied in town planning and architecture, but has recently been taken up by object-oriented software designers [5]. A key aspect of its original impulse is that it enabled designers of buildings to capture that “quality without a name” which could be felt in buildings which worked for their inhabitants, but which was hard or even impossible to formalise. Common features of buildings which have this quality are identified as resolving a particular problem, and this is presented as a pattern — a guide to implementation that must be interpreted by individual designers, but which will have certain invariant features. Patterns occur at different levels within an artefact, and a pattern at one level will imply a number of patterns at a lower level to complete it. Thus a pattern for a room will imply patterns for entrances, walls, windows, ceiling height and so on. These linked patterns provide an informal grammar for good design in the domain — a Pattern Language.

The design of usable software is a particularly appropriate domain to apply patterns as the subjective feelings of users about well designed software appear similar to the “quality without a name” that is sought in architecture [7]. In interfaces we call it “usability”, “conviviality”, “engagement” and so on. Several usability design patterns have recently been proposed [3, 6] and at least one extensive pattern language has been written [8].

The development of the knowledge and skills required to write usability pattern languages holds much promise for meeting the educational needs of interface designers. We have begun to apply these ideas in our undergraduate and postgraduate teaching at the University of Brighton.

PATTERNS IN USABILITY DESIGN TEACHING

We are using three different but complementary approaches to incorporate pattern language. The first, teaching *about* Pattern Language is really teaching about design and involves making connections between software design and the debates on design theory that have gone on since the 60's in the fields of architecture and industrial design [2]. This is unfamiliar territory in software engineering syllabuses but is an area which deserves to be presented more centrally. The recent advent of Tidwell's substantial HCI pattern language [8], has made it possible to take the second approach, teaching *through* Pattern Language, by setting students practical exercises in using a pattern language in the process of design. We have begun small-scale exercises using Tidwell's patterns and informal findings are that both the quality of discussion within design teams and the product were considerably improved. We intend to do more work in this area and attempt to verify this impression more formally.

We explore here a third approach, taken with a small group of Third Year students of User Centred Interaction Design, to encourage students to discover and define interface patterns for themselves. The rationale for the approach was that discovery of Patterns by students would avoid misconceptions about the status of any published patterns which they were subsequently asked to apply: it would be a process of demystification. The Pattern approach was explained in a lecture and the students were directed to a range of relevant Web sites and readings. They were then asked to identify and catalogue four patterns. Although they claimed to have understood the Patterns approach, when they came to attempt the task themselves they found it extremely difficult and asked for guidance on the process of pattern discovery. We suggested two strategies: modifying existing guidelines and drawing on personal experience. Patterns can be thought of as richer versions of the guidelines listed in every HCI text book, e.g. be consistent, provide feedback and so on. Any student choosing this approach simply had to find such a list and add some "meat" to it. None of the students chose to do this: they explained that starting with guidelines didn't make the task of linking the specific and the general any easier for them.

Instead, they used the second approach, examining their own experience of computing and building a pattern from some feature which seemed an example of either good or bad design. Good design is notoriously difficult to spot since by definition it does not call attention to itself. Bad design, on the other hand, makes its presence felt. All the students' examples were derived from bad rather than good design. They drew on their experience of breakdowns caused by inappropriate design decisions: aspects of their software which frustrated them in their own work. The eleven students presented 37 different suggestions for patterns. Surprisingly, despite our extended preliminary discussions, only a few of the suggestions approached the status of true patterns. The majority of suggestions stayed at the level of ungeneralised examples. For instance, one student described his frustration at his system halting a reboot to tell him it has found a non-system floppy disk in its drive. Another described the sequence of operations he had to perform during start-up to rid the screen of a series of dialogue boxes informing him that his printer was unconnected. Another mentioned the need on his system to confirm a print command, which typically appeared on the screen as he was already on his way to the remote printer. It was only when we discussed the examples in class that it became clear that each of these problems would be solved if designers had used a pattern such as "Don't interrupt operations with trivial problems".

CONCLUSION

Although they understood an exposition of the Pattern Language approach, even Final year students found it difficult to generalise from examples, a high level cognitive skill. A two-stage process is needed if generalisations are to be made, with individuals presenting examples from their own experience from which can be identified. This confirms Alexander's own feeling that identifying good patterns is as hard as doing nuclear physics, but doesn't discourage us from planning further experimentation with Pattern Languages for teaching interaction design.

REFERENCES

1. Alexander, C., Ishikawa, S. & Silverstein, M. A Pattern Language: Towns, Buildings, Construction. Oxford University Press, 1977.
2. Cross, N. Developments in Design Methodology. John Wiley, Chichester, 1984.
3. Erikson, T., The Interaction Design Patterns Page. Available at http://www.pliant.org/personal/ Tom Erickson/InteractionPatterns.html.
4. Erickson, T. Interaction pattern languages: A *lingua franca* for interaction design? *Invited talk presented at the UPA Conference 1998* (Washington, D.C., June 1998) available at http://www.pliant.org/personal/ Tom_Erickson.
5. Gamma, E., Helm, R., Johnson, R. & Vlissides, J. Design patterns : elements of reusable object-oriented software. Addison-Wesley, 1995.
6. Griffiths, R. Brighton Usability Pattern Collection available at. http://www.it.bton.ac.uk/cil/usability /patterns/.
7. Pemberton, L. and R. Griffiths. The Timeless Way: Making Cooperative Buildings with Design Patterns. In Proceedings of CoBuil98, Speinger Verlag, 1998.
8. Tidwell, J., Common Ground. available at http://www.mit.edu/~jtidwell/common_ground.html.