Chapter XXVII
Pattern Languages for CMC Design

Dan Dixon
University of the West of England, UK

ABSTRACT

Three decades ago the concept of pattern languages were introduced in the field of architecture and they have since become widely used in object-oriented programming and HCI. However their use in computing is divergent from Alexander's original goals on two main points. Firstly, they were largely intended to describe the spaces formed by or for human activities and events. Secondly, they were intended as a way for professionals and lay people to communicate whilst designing buildings. This chapter suggests that the socio-technical design of social software should rediscover both these principles, firstly in a fuller appreciation of the wider human angle, and secondly in the participative design approach. Indeed, a pattern language approach within a socio-technical framework seems the ideal way to design the next generation of computer-mediated communication applications, as it will do so in a social context and in partnership with end users.

The power to make buildings beautiful lies in each of us already.

—Christopher Alexander

INTRODUCTION

Over the last decade many new forms of computer-mediated communication (CMC) have appeared. These have progressed well beyond the original forms of, largely interpersonal, communication that appeared with the early internet; email, usenet and internet relay chat (IRC). Large scale social networks, social bookmarking, wikis, media sharing, instant messaging and now mobile applications are all reshaping our understanding of CMC. Many of these new applications involve large-scale, net-

worked interactions and the people who use them are now very different from the academics and engineers who originally built them.

In software design there is a process gap between understanding the overall objectives and having a well specified system that can be handed programmers to build. In software product management terms, this often gets called the 'fuzzy front end' of a project. At the beginning of any product development, the team involved, will often a lack an understanding of what the software product needs to be and lack a clear process on how to move from what are usually business or organisational level goals to the level of detail needed so that developers can create the correct software (Khan, 2004). Different projects and organisations tackle this in different ways. Often they will tend to rely on experts who have deep tacit knowledge in specific application areas and experience of working on many projects. This approach locks knowledge into these individuals, makes them critical to the process and isn't necessarily conducive to a participative approach.

In most cases this gap is currently filled by techniques like requirements gathering, functional specification, use cases or user stories. These techniques are very useful if the end product is well understood but they are better used as part of the specification phase of a project. In systems focused on single user applications or in well understood domains they may be sufficient to bridge the gap between the 'fuzzy front end' and software design, but aren't sufficient to plug the gap when we are building systems for computer-mediated communication. These single-user focused approaches to specification try to understand interaction as atomic tasks, and focus only on interactions between the user and the system. The sociotechnical systems we are looking at are inherently complex systems and because they are complex, they also need to be understood on a macroscopic scale through their emergent properties. Reductionist approaches, like the ones described above, will not work and so a technique that looks to emergence is required.

The architect Christopher Alexander was tackling similar problems and introduced the idea of patterns and pattern languages as a way of creating a participative design process. Patterns are a way of recognising and describing approaches and structures that are encountered repeatedly in a discipline. He described patterns through a three part rule; a relationship between contexts, problems and solutions (1979). These ideas although not being used commonly in architecture have been adopted widely in computing. In architecture these patterns are common parts of buildings or the ways that urban areas are laid out, in software development they are common ways that objects interact, and in HCI they are common site architectures or interface components.

Architecture has a lot in common with software design. Across both disciplines the designers are trying to create artefacts that fit into and enhance people's daily lives. Sociotechnical systems theory is, a now very established, approach to understanding how people and organisations relate to technology. The idea is that there are inter-linked human and technological systems involved in the use of any technology. Sociotechnical design was an approach that used these ideas as a basis for systems development. These ideas that started with investigations into how people use very physical machinery now make even more sense in an age of online social networks (Pasmore, Francis, Haldeman & Shani, 1982).

Pattern languages using a sociotechnical background are a possible way of bridging the divide between high level objectives and the lower level specification of any CMC system. Pattern languages can be used as a shared communication and collaboration tool between the technical professional and the lay person and they can be used to create an intermediary design artefact—an application pattern language for a product or system. This is exactly the way that Alexander intended pattern languages to be used in architecture, a very high level way of describing a building before drawing up any detailed blueprints. Additionally, this suits modelling the complex systems of computer-mediated communication as it is concerned with documenting and understanding the emergent properties at different scales, not delving into details.