Analysis of Design: An exploration of Patterns and Pattern Languages for Pedagogy

Sally Fincher, Computing Laboratory, University of Kent at Canterbury, Canterbury, Kent, England
+44 1227 824061
S.A.Fincher@ukc.ac.uk

1. ABSTRACT

This paper describes and delineates patterns and pattern languages from different knowledge domains, and attempts to generate an understanding of why they are a distinctive and powerful way of sharing knowledge. It surveys several examples within the genre of Pattern Languages and analyses the unique characteristics of the form, with reference to the instances surveyed. From this basis, it makes the argument that a Pattern Language of Pedagogy is possible and achievable and that for historic and disciplinary reasons CS is singularly well positioned to create such a tool and a particularly fertile ground for its use.

1.1 Keywords

Pattern, Pattern Languages, Pedagogy

2. INTRODUCTION

Patterns and Pattern Languages are very important at the moment in Computer Science. They have been developed to allow expert practitioners to capture and share design expertise and the powerful ways in which they do this have led to their success. However, with the proliferation of applications of the pattern form to areas outside of the originating domain questions such as "What are patterns?" and "What do patterns mean?" now have multiple and often divergent answers.

For the purposes of education, it would of great utility if we could discover and develop ways to exchange knowledge about the design of effective learning environments. Teaching and learning activities have a long history and successful models are often imitated and preserved (apprenticeship, mentoring, problem-oriented project work for example). Some people, influenced by work on software patterns, have worked towards creating pedagogical patterns for classroom use. However, this paper argues that (separate from and additional to this use) the pattern form is singularly well adapted for the sharing of good practice between practitioners, and that this has great scope for the future of teaching and learning within CS.

Section three briefly examines the genesis of patterns and pattern languages, and places the existing ideas around pedagogical patterns into that historical context. Section four presents an analysis of the constituent parts of pattern languages, in an attempt to generate a working understanding of their construction. Section five looks at current pedagogical patterns work, together with other initiatives which are working towards similar ends from different perspectives, and argues that all point to the concept of a pedagogic pattern language. Finally, two possible ways of using pattern languages to share expertise in the design of teaching and learning environments are identified

3. A SHORT, HISTORICAL, SURVEY

3.1 Alexandrian Patterns & Pattern Language

One of the interesting things about Patterns and Pattern Languages is that there is a single first-use instance of the genre. Unlike, for example, detective fiction, where one can say "Edgar Allen Poe invented this type … and Arthur Conan Doyle pioneered that approach … and Raymond Chandler added that aspect …" for Patterns it all starts with Christopher Alexander's works *The Timeless Way of Building* (Alexander, 1979) and *A Pattern Language* (Alexander, Ishikawa, Sliverstein, 1977),. In *A Pattern Language* particularly, he identifies Patterns and creates an organising principal for their use.

Alexandrian Patterns are concerned with architecture and buildings. Individually each pattern "… describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem" (Alexander et al,

Reference for this paper: *Journal of Computers in Mathematics and Science Teaching: Special Issue CS-ED Research,* 18(3):331-348, December 1999.

1977, p. *x*)[1] The Patterns are constructed to convey information about the design of architectural artefacts from experts to a wider audience (users, or inhabitants) in the least constraining way possible - so that it does not matter if the design is realised in mud, straw or bricks; only that the fundamental principle is embodied. Each problem/solution pair, which constitutes the Pattern is presented in a common format that consists of:

A PHOTOGRAPH	showing an archetypal example of the pattern in use
AN INTRODUCTORY PARAGRAPH	which sets the pattern in the context of other, larger scale patterns
THE HEADLINE	an encapsulation of the problem (one or two sentences)
THE BODY	of the problem (this can be many paragraphs long)
THE SOLUTION	the heart of the pattern, always stated in the form of an instruction
A DIAGRAM	shows the solution in the form of a diagram
A CLOSING PARAGRAPH	shows how this pattern fits with other, smaller patterns

Figure One: Alexandrian Pattern Format. The sections and descriptions are from Alexander (Alexander et al, 1977), the comments in brackets mine

Each pattern as well as containing these common elements is presented in a consistent typographic style, making it easy to find the same part (the headline, or solution for example) across several patterns. Individually, these are powerful encapsulations: like an entry in an encyclopaedia or dictionary, they convey a large amount of information, usefully compressed.

However, the individual patterns are only part of the story. The other component of the book, the "Language" of the title, is at least equally important. "Language" for Alexander has two senses. The first sense is that we all share a design language that his work merely articulates. For example, in very cold countries, it is common to put planks (or rocks, or whatever is to hand) on the roof to hold back the layer of snow in winter, so that it does not fall off in one piece, endangering passers-by. This is not written down anywhere, no-one is explicitly told to do this, it is "obvious": it is part of the local vernacular design vocabulary - and cannot exist in countries where there is no snow. This is why, for Alexander, there is no single Pattern Language, only a concatenation of Pattern Languages.

The other sense of "language" is of an organising principle which facilitates the use of patterns. Just as a dictionary is (normally) organised on an alphabetic principle, thus allowing the user to find the required word easily, so the "language" of the patterns organises them to facilitate access. Of course, it's not quite as simple as with a dictionary. Partly this is because the organising principle for design components is not as straightforward as an alphabet. Partly it is because the language is not totally separate from the patterns, but is articulated (realised) by the relationships in which they stand to each other. Mostly, however, it is because the aim of the "language" is generative: that is to say the *language* aim of the work is to engender the facility to combine and re-combine patterns (depending on specific situations, desires and constraints) to generate satisfactory solutions to given problems. Whilst not a precise analogy, this has the flavour of the generative power of natural languages where pieces at different levels—words, verbs, metaphors, sonnets, expositions—can be combined and re-combined to generate different artefacts in different circumstances.

3.2 Software Patterns & Catalogue

Almost 20 years after Alexander's ideas were first published, a second book on patterns and their use was written: *Design Patterns: Elements of Reusable Object-Oriented Software* (Gamma, Helm, Johnson, Vlissides, 1994) (this book is almost always referred to by the collective name for the authors—the "Gang of Four"—or GoF). This time, the subject domain was not architectural design and the building of houses, but software design and the building of systems: specifically, object-oriented systems. Unlike *A Pattern Language*, *Design Patterns* was not without precedent. It follows Alexander, but adapts the genre for a new domain. GoF patterns "describe simple and elegant solutions to specific problems in object-oriented software design" (Gamma et al, 1994, p. *xi*). The aim of these patterns is not to share software design knowledge between experts and a wider audience (users), but to share knowledge between experts in the OO paradigm: "design patterns provide a common vocabulary for designers to use to communicate, document and explore design alternatives"(Gamma et al, 1994, p.352). Following Alexander, each pattern is expressed in a common format, which in this case consists of

[1] This is a technical definition of a term; the fact that it adopts a word which has a common meaning has led to confusion in subsequent usage and development of understanding in this field.

NAME	(this is considered to be important because it provides a "handle" on the idea the pattern contains and allows a shorthand common vocabulary between designers)
INTENT	what does the pattern do
ALSO KNOWN AS	alternative names
MOTIVATION	A scenario that illustrates the problem
APPLICABILITY	The situations in which the pattern can be applied
STRUCTURE	A graphical representation using a specific system of notation
PARTICIPANTS	The constituent objects and classes
COLLABORATIONS	How the constituents collaborate
CONSEQUENCES	
IMPLEMENTATION	
SAMPLE CODE	
KNOWN USES	
RELATED PATTERNS	

Figure Two: GoF Pattern Format (Gamma et al, 1994). The sections and descriptions are from Gamma, the comments in brackets mine. Note that they provide no explanation for some sections.

The patterns are physically much longer than Alexander's so, although they too have a common typography, this is less readily apparent. Also, as can be seen by the headings, they include a much greater amount of very concrete information, graphically detailing abstract structure and even particular structure by including code fragments (which might be thought of as providing an Alexandrian solution in terms of what bricks to use, although there is an interesting alternative exploration of the function of this section. See Section 4.5, below).

GoF are very clear both about their debt to Alexander and about the "linguistic" limitations of their design patterns. The explanation given for this is twofold. Firstly, the fact that the history of software design is much shorter and more localised (within a community) than the history of buildings. From this they assert that there simply isn't as much human knowledge (either explicit or implicit) about how to do it and what makes it good. Secondly, they don't seem to believe in the generative power of the organising principle (the "language"). They call their book a "Catalog" and describe it as "just a collection of related patterns; we can't pretend it's a pattern language". (From the outside there seems to be another reason. *A Pattern Language* contains 253 patterns, *Design Patterns* 23. It may be that you just don't need a generative "language" for something that small.) What they do have instead is a categorisation structure based on conceptual phases of the design process, so patterns are Creational, Structural or Behavioural.

3.3 Pedagogical Patterns for Teaching Object Technology (PPTOT)

Patterns have "taken off" in Computer Science in a way that they never did in Architecture, and there are now many web pages, mailing lists, books and conferences that exist to support, discover and discuss patterns. One of the spin-offs from the GoF impetus has been the creation and collation of "pedagogical patterns for teaching object technology" (Pedagogic Patterns, 1998). This collection has been made over time (the project originated at OOPSLA'95 (McLaughlin, Sharp, Manns, Prieto, 1998) and has continued through workshops held at "Pattern Languages of Programming" conferences and the like) and is a direct-line descendent from GoF software patterns; their format is consequently familiar. They make no reference to Alexandrian patterns.

NAME	pattern name
INTENT	what the instructor wants to teach, or avoid, or...
IDEA	how this pattern can achieve the INTENT
MOTIVATION	describes why the instructor achieves the INTENT with this pattern
APPLICABILITY	circumstances in which pattern is most useful, in the opinion of the pattern author and others who have used the pattern
CONTRAINDICATIONS	when not to use the pattern
STRUCTURE	description of the pattern's elements
CONSEQUENCES	what has been seen to occur when this pattern has been used
ISSUES TO CONSIDER	the pragmatics of using the pattern

CULTURAL DEPENDENCIES	issues which may make this pattern less useful for a particular culture
RESOURCES NEEDED	the resources which are needed to implement the pattern
EXAMPLE INSTANCES OF THIS PATTERN	specific instances in which the pattern has been successfully used
RELATED PATTERNS	other patterns which are related to this pattern

Figure Three: PPTOT Pattern Format. The sections and descriptions are from (McLaughlin et al, 1998)

There are 43 patterns in this collection, written by many separate authors. Although it is not clear, they seem to have been generated because there were people around who were used to the idea of patterns (in the software sense) and thought to search for them in another domain. Consequently, they form a much less coherent whole than the other two works. Most obviously in this respect they lack a cohesion of scale, encompassing material from the use of learning theory concepts to improve the effectiveness of lecture sessions (No. 9: Gagne-Ausbel Pattern of Lecture (GAP) Pattern) to a precise method for teaching the use of accessors and mutators for accessing an object's private data (No. 12: "What did you eat for breakfast?" Pattern). (Pedagogic Patterns, 1998).

This survey is partial, covering examples which are of especial interest to CS education. However, the chronological development of patterns and pattern languages in this domain does not expose the features which distinguish the pattern form from any other, for example a library or similar collection of related resources. In the next section an analysis of the form is presented which aims to abstract and characterise the elements which make up a pattern language.

4. NECESSARY CHARACTERISTICS OF PATTERNS AND PATTERN LANGUAGES

There exists no general definition of what constitutes a pattern or the genre of pattern languages– although there are many actual examples. This has a number of consequences. It makes it very difficult for a pattern "outsider" to understand what is meant by the terms. It means that when patterns are formally taught, often what is learnt is a specific pattern form presented in a tightly specified way. It is difficult for the learner to abstract from these specifics, and no distinguished abstraction is presented. It means that when patterns are written, the intent is often confused with the form: the problem with the solution.

In an attempt to arrive at an appropriate and useful general definition, examples of pattern languages were analysed for essential commonality. The aim of this analysis was to be able to arrive at a description which could be used in a meaningful way; to be able to say "If it doesn't have this, then it's not a pattern (language)". In this analysis of the form, four major elements (and one minor one) which characterise patterns and pattern languages were identified. The major elements are: Capture of Practice, Abstraction, Organising Principle and Value System; the minor element is Presentation.

4.1 Capture of Practice

It can be easy to mistake a codification of a specific piece of practice for a pattern. This statement is so true that it is tempting to echo E.M. Forster's (Forster, 1927) lament "Yes—oh dear yes—the novel tells a story". Indeed, a pattern must contain a specific example of practice, because patterns aim to convey knowledge about design of *environments* (be they architectural, software or pedagogic) not "design" in the abstract. However, the piece of practice, the example that demonstrates and illustrates the application of the design principle, is only one constituent. Bayle et al (Bayle, Bellamy, Casaday, Erickson, Fincher et al, 1998) discovered within the context of work on User Interface Patterns that it was relatively easy to observe phenomena which could be put into a pattern-like form, but this act of capture was not sufficient.

The missing factor, the additional requirement which turns practice into patterns is an intentional and creative process on the part of the pattern author(s). Undertaking this process necessitates active consideration of the other three major elements. Capture of Practice is a necessary element, certainly, but by no means sufficient. Ultimately, it is the least part of the achievement.

4.2 Abstraction

There is never a single way to build a door, or write a loop, or teach a class. If patterns are really a format for capturing exemplars of *design* then it is not enough that they simply capture practice, the practice captured must be illustrative of a *successful* way to solve a given problem. The characteristics which make the design a success must be abstracted from the example; the "good way" is then made understandable and therefore transferable to other practitioners in other situations. How this level of abstraction is achieved is difficult to observe. Alexander speaks little of this process, although he does say that the observation of common structures in many separate and disparate environments prompted his team to think that the

commonality of their design reflected something fundamental in the way people like to live in buildings. Alexandrian patterns are even annotated with a "star rating" (zero, one or two stars) which indicates how closely the authors believe the pattern in question approaches this truth. Other pattern authors also claim that abstraction is a product of observing a number of separate incarnations of a single solution, and the PPTOT group have institutionalised this (for their project) by requiring that no practice can be a pattern unless three examples of its use can be found (the so-called "Rule of Three"). For pedagogy, this is problematic; much teaching occurs behind closed doors, in a very private environment, and is simply not available to be observed in the same way as a roof-type or even an algorithm.

Equally, as is demonstrated in many teaching and learning experiences, it is not necessarily an easy thing for learners to apprehend the root of a common problem from a collection of solutions (implementations), and it seems unlikely that application of the Rule of Three can be used as a substitute for the hard process of an expert abstracting commonality (with examples) for the benefit of others. It is possible that the process works the other way up; that from accurately capturing practice, abstraction is made possible. For example, Astrachan (Astrachan, Berry, Cos, Landon, Mitchener, 1998) notes a common problem for students in processing sequential data and abstracts a patterned solution: One Loop for Linear Structures.

Abstraction also serves a second purpose, that of cohesion of ideas. Practice can be captured at any scale, but it is the combination of capture and abstraction that makes the presentation of the ideas coherent. Lakoff (Lakoff, 1987) presents an example of this coherent use of abstraction in regard to the Linnean taxonomy of botanical classification - Genus, Species, Sub-Species, Variety.

An oak tree (for example) can be categorised at any level. However, in folk-classification (as opposed to scientific) Lakoff (quoting studies by Brown and Berlin, p.35) notes that the most commonly used (and by extension, the most significant) name and reference is at the level of abstraction that corresponds to the genus level ("oak") rather than the life-form ("tree") or variety ("white oak") levels. Whilst an interesting observation in it's own right, the more interesting point is that Linneaus actively used folk criteria for the genus level of abstraction which characterise the most readily apprehended (and used) criteria in "the real world".

This is a concept equally important in OO, Booch's (Booch, 1994) codification of "key abstractions", notes that there are levels of categorisation which are more significant in the problem space, and useful to the solution design, than others. He, too, suggests that these might most effectively be identified from actual usage "if the domain expert talks about it, then the abstraction is usually important".

What has been noticed here is that some categories of abstraction are more basic, more meaningful to human beings in their relationship with the world than others; that is the level of abstraction that good patterns seek to embrace.

4.3 Organising Principle

As we have seen, patterns do not exist by themselves, but within a framework: a catalogue or language. In a catalogue, the power of the collection resides in the material collected. The index, or finding aid or other system of organisation is of secondary importance, it is simply a mechanism to get to the information. In a dictionary, encyclopaedia or thesaurus, the power resides as much in the arrangement of material, in the power that the organising principle confers to it, as in the individual entries themselves. The solitary definition of a word is useful, but much more potent in the context of a dictionary. The Organising Principle of a Pattern Language has a similar gestalt power; the language captures not only the pieces of design, but the shape of the whole into which the pieces fit.

The Alexandrian organising principle is scale. *A Pattern Language* recognises the impossibility of providing a complete solution ("Here's the plan for the house/street/city you want to build") so presents many small, transferable solutions arranged in categories of scale, from "city-relevant" to "house-relevant". Consequently, if I am building a house, not a whole street, I have an obvious entry point to the most appropriate level of patterns. The boundaries for these categories, however, are not hard and Alexander provides pointers, both up and down, through the levels pointing to larger-scale patterns to which a given pattern is contributory and to smaller-scale patterns on which it rests.

The GoF framework is much simpler, residing on the applicability of their solutions to different functionality in the design process (Creational, Structural or Behavioural).

The PPTOT collection has neither order, nor organising principle: instead, four separate "indexes" are provided, each on a different axis: A *Learning Objectives Index*, a *Teaching/Learning Element Index*, an *Alphabetical Index* and an *Author Index*. It is clear that these indexes have been super-imposed on the collection at a later date, and that they have had no impact at the time of creation. (However, it should be noted that PPOT is a work in progress, and recent work has turned towards structure and taxonomy. Although full details are not available, it would seem that this, too, is based on scale from "a technique pattern aimed at explaining a particular OO concept" to "a complex structure pattern built from smaller fundamental units, such as lectures and exercises" (McLaughlin et al, 1998)).

A potentially interesting approach, which has not so far been adopted within the genre is one espoused by Jacobson (one of the original contributors to *A Pattern Language*) in a later work (Jacobson, Silverstein, Winslow, 1990). In this work the organising principle for good design is the balance achieved on various axes of contrast. Six axes are identified with respect to architecture with "good" design representing an equilibrium along and between these scales. A similar methodological approach (Wildermeersch, 1997) has been taken in an attempt to capture (and improve) the practice and experience of working within project-oriented groups; a domain with more obviously pedagogic potential.

4.4 Value System

Design is a purposeful, value-laden activity. Good design encompasses values which are of importance to all the communities (or audience) for which the artefacts are intended. For the purposes of patterns, we can define three audiences: users, other designers and "society".

- **Users**: Values which are important to users are those embodied in the artefact itself. This is exemplified in the arts-and-crafts motto "Have nothing in your house which you do not know to be useful or believe to be beautiful". If an object is difficult to use, or ugly, then it fails in its purpose. It is disfunctional.
- **Other Designers**: Whilst fellow professionals might be expected to appreciate the values of the user community, they also hold another set of separate importance. These professional values encompass design notions of "elegance" and "simplicity". An appreciation of the economy of immaterial processes (such as maintainability, factory production or services) also count here.
- **Society**: Design also addresses ideas of value which are societally constructed. This is a more difficult (and difficult to observe) constituent of patterns and pattern languages. This is because this level is not directly addressed by patterns, although they are created against its backdrop. For example, the Alexandrian pattern 178, describing Composting Toilets, (Alexander et al, 1977) encompasses a far more wide-ranging set of values than a thing designed to do a job efficiently—that is usefully—and/or beautifully. (These values are not necessarily commonly held – but they would have to be, if everyone was to own one. They are of the societal level). Patterns are described and presented as internally consistent system of elements that are good in themselves and in relationship to each other. Any further set of values is extrinsic to this ordering, and therefore not described within a pattern collection. This notion of extrinsic validity is analogous to IQ tests which are internally consistent, valid and predictive (as are measurements of height). Their value, however, is neither measured nor contained within the application of the test but is determined by a separate, external system. A society which values high IQ (or tall people) gives a separate – and extrinsic – meaning to the results. The fact that (to continue the example) *A Pattern Language* doesn't explicitly incorporate these values doesn't mean that the patterns weren't created with reference to them. It may be that there is no need for these values to be expressed in this way. Just as Fine Art reflects the values of the society in which it is created, so may design: "The best paintings often express their culture not just directly but complementarily, because it is by complementing it that they are best designed to serve public needs: the public does not need what it has already got." (Baxandall, 1972).

Alexandrian patterns have three audiences: architects (other designers) and the inhabitants of the buildings (users) are explicitly addressed, society is implicitly addressed. PPTOT patterns have two audiences, teachers (other designers) and the recipients of teaching (users). GoF patterns have but one audience – other designers – and reflect a single system of purely professional values. Christopher Alexander (whilst acknowledging his lack of expertise in the domain of software) has expressed his opinion that software patterns (and, I think by extension patterns based on that system) are not patterns in his sense but "a neat way to capture a bunch of interesting ideas" (Alexander, 1996). He does not explain why he thinks this is the case. It may be the constraint of their single value system.

4.5 Presentation

The most obvious single point of difference between the format of Alexandrian patterns and the others examined is that the name of the pattern is not part of the Alexandrian template, indicating that the authors did not perceive it as an important element. This is quite contrary to the experience of regular users, who believe names to be extremely important, for two

reasons. Firstly, patterns are only ever referred to by their name. Being able to refer to a pattern by its name, without having to explain the underlying detail is a valuable and powerful tool. Secondly, (and perhaps dependently) because names are used as synecdoche the accuracy of the name in conveying the purpose of a given pattern is held in great esteem.This Alexandrian lack of attention to naming has not been continued and the name holds a place of considerable importance in almost all other patterns and pattern templates.

The common part of presentation, and a very strong one, is the inclusion of a concrete example of implementation of the pattern. This is not the textual description which in almost all cases forms the body of the pattern. In Alexander, the photograph conveys this example of implementation, and it comes first. In GoF patterns it is the code sample. The importance of this part of the pattern template is not immediately obvious, but is difficult to stress too much. I believe that the purpose of these components is to sensitise the reader to the application of the pattern.

In looking at the photograph, a reaction is invoked. The intention is that the reaction is favourable—"Wow, that's good. I'd like to live there"—and from that point the reader is sensitised so that the information that the rest of the pattern contains becomes more accessible, more useful in a specific implementation.

The code sample in GoF is buried quite low in the pattern template, but it has been observed that "The example actually explains the solution a lot better than does the solution description" (Gabriel, 1996). I think that this, too, is a reflection of the sensitising function of the concrete. The PPTOT template buries the requirement for the concrete by requiring "Example Instances of this Pattern" which is variously interpreted.

5. INITIATIVES FOR SHARING PEDAGOGIC EXPERTISE: POINTERS TO A PEDAGOGIC PATTERN LANGUAGE?

There are several projects being undertaken at the present time working to capture the "private practice" of the classroom and make it more widely available and usable. Whilst none of these projects call themselves pattern languages, their work does display some of the constituent parts. Two are of particular interest in this context.

- **The On-Line Computer Science Teaching Centre** (CSTC) (On-line, 1998) is based on the idea of peer-review of practice as an indicator of quality, in a manner analogous to the peer-review of research papers. The concept of peer-review is well understood in the academic world and has been used for many decades. It is a successful solution to the specific problem of assessment of contribution. It relies on a value-system shared by the community to which the papers are presented. At present, there is no such shared value-system for teaching and learning materials, practices or environments. If this initiative can identify one (or central parts of one) within the community, then CS Education will have taken a large step forwards. It may even be that such a value-system could be *created* through the iterative process of developing acceptable criteria for such peer review.

- **Effective Projectwork in Computer Science** (EPCoS) (EPCOS, 1999), is undertaking an endeavour closer to the pattern approach in that they have developed a template to facilitate the transfer of project work practice between institutions. Use of this template attempts to abstract practice to make it context-independent or explicate it (rather like printing the ingredients on the label of a tin) to make it context-transparent. This process of explication and abstraction is called "bundling" and the resulting products "bundles" (EPCOS, 1999) The format for a bundle is:

A. NAME OF BUNDLE	
B. BUNDLE HEADINGS	
BUNDLE DESCRIPTION	"This is what it is"
BUNDLE PURPOSE	"This is what it does" (Sometimes called aims and/or objectives.)
BUNDLE BENEFITS	"You should adopt this, because ..."
BUNDLE PROBLEMS	"Sometimes it blows up, because ..." (We allow projected, as well as discovered, Benefits & Problems.)
C. CRITICAL CONTEXT	
CONTEXT DESCRIPTION	"Where it fits"
CRITICAL DEPENDENCIES	"It doesn't work unless... or if ..."
CRITICAL ADJACENCIES	"It probably works better if ..."
D. EVALUATION	

Reference for this paper: *Journal of Computers in Mathematics and Science Teaching: Special Issue CS-ED Research,* 18(3):331-348, December 1999.

SUCCESS OF TRANSFER	"You'll know it's worked for you if these things happen and these other things don't" (or "How to measure benefits")

Figure Four. EPCoS Bundle Template (EPCOS, 1999) Parentheses in the original.

This template is then attached to specific material (of varying nature - sample handouts, code whatever is being discussed) which concretises the information and allows for easy transfer.
It is obvious that bundles are not patterns. The only purpose of a bundle is to describe practice to a point where it can be transferred from one context to another. All EPCoS bundles have been created (by many separate authors) solely with this purpose in mind and, like all such collections of material, some bundles are better than others.

EPCoS bundles and PPTOT patterns would seem to be pretty similar, but there is a single strong difference. EPCoS bundles have been created with the (Alexandrian) objective of transferring good design from one context to another. PPTOT patterns were created *as* patterns with the (GoF) objective of encapsulating good design. This difference is critical because the point of the *use* of patterns is to invoke a reaction in their audience. The patterns themselves must convey information, must be information-dense to allow the pattern-user to understand them and construct their own specifics from them. But the desired consequence of using the patterns is not the transmission of information but the invocation of a reaction. Without an idea of the purpose for which the patterns are being created, without an idea of the uses to which they are going to be put (and hence what reaction they are intended to invoke) the creation of the patterns becomes an information-gathering exercise, rather than an attempt to share something "good".

- **Pedagogic Patterns for Teaching Object Technology** It is clear that whilst PPTOT identifies closely with the concept of pattern languages, the PPTOT patterns are different from both the primary examples of the genre. Although they follow the GoF development, they have little coherence of abstraction, omit the consistency of the value system and do not require the sensitising example in their template. There is no large body of users for PPTOT patterns, as there is for GoF ones, and this may be attributable to this lack of key characteristics which make the form uniquely useful. However, it may be that, looking at the genre of Patterns and Pattern Languages, that the GoF format is not the best adapted for pedagogic requirements.

If Architecture has a long history and vast constituency (almost everyone lives with buildings, however well or poorly designed) then equally teaching and learning has a long history and almost everyone has learned or been taught something (however well or badly this was accomplished). The fact of this extensive history and body of knowledge overcomes the GoF objection that there simply isn't enough experience to construct a "language". Also, teaching and learning can be thought of as a point of combination of the institutional and social worlds (and their value systems) in similar way that architecture combines the physical and the social worlds. Pedagogy is quite as appropriate a domain as architecture for the discovery and collation of Alexandrian-style patterns. A Pedagogic Pattern Language could be a way to capture fundamental solutions to everyday problems of teaching. It could be a work that facilitated the design and construction of effective educational environments and, eventually, it could even be a work that shared educational expertise with the end user; to the point of allowing learners to construct learning environments for themselves.

One of the surprising side effects encountered in EPCoS is that for effective transfer, the practices which bundles capture must be small-scale. The users of the bundles are in general unable (or unwilling) to change the entire context of their practice. Just as few people are able to say "I shall build a four bedroom house in an attractive tree-lined residential area, for less than half the market price" so few teachers are able to say, "I shall teach a class on object-technology to 15 intelligent, motivated, students twice a week for the next 6 months"

This provides another indicator of the potential kinship between pedagogic and architectural pattern forms. If, for effective transfer of practice, bundles must be small-scale; then for effective transfer of design pedagogic patterns must be small-scale. This points to the necessity for pedagogic patterns to espouse a combinatorial organising principle, in the Alexandrian mode.

6. SUMMARY AND FUTURES

The genre of pattern languages started with Alexander. GoF adapted it to a specific domain and purpose. From that adaptation, PPTOT adopted the GoF style and format. CSTC and EPCoS (in pursuit of other ends) display characteristics of form which are more akin to Alexandrian aims. The Alexandrian pattern form is potentially well adapted to the sharing of design knowledge about teaching and learning environments, and, because of the history of design patterns in the discipline, CS is particularly well placed to utilise this.

Reference for this paper: *Journal of Computers in Mathematics and Science Teaching: Special Issue CS-ED Research,* 18(3):331-348, December 1999.

It is a difficult and dangerous thing to try and predict trends. However, current usage and interest suggests that we have only scratched the surface of the potentiality of patterns and pattern languages. Here are two possibilities for how they could be used in a vision for the future of CS Education.

- "Pattern" has replaced OO as the buzzword of the nineties. This means that even though there is little shared knowledge of what patterns are or how they might work for pedagogy, software design patterns have been widely adopted by Computer Science Educators in their teaching (of OO systems in particular and, more recently, of basic programming concepts). With such widespread adoption of a tool formulated for a separate purpose, a carefully developed, tailored Pedagogic Pattern Language could be expected to be very well received and generate significant momentum.

- Given the practitioners' habit of using design patterns as a shorthand vocabulary between expert and expert, a set of behaviours and expectations for the form already exists within CS. Some teams already keep a "library" of the software patterns used in a given project so that they can quickly grasp and share design features with each other—and other projects (Brown, Malveau, McCormick, Mowbray, 1998). This use could be extended to capture the local, vernacular design languages of varying pedagogic environments—departments in research universities, distance education activities, CS1 courses, project based courses, web-based courses, community night schools—and facilitate the sharing of that expert understanding to other practitioners both locally and across time and space.

Why should we bother?

Because we already *have* such expertise. The pity is that we neither capture it nor share it.

7. REFERENCES

Alexander, C, Ishikawa, S, Silverstein, M, *A Pattern Language: Towns, Buildings, Constructions*. OUP 1977

Alexander, C *Patterns in Architecture* Presentation at OOPSLA'96 University Video Communications 1996

Alexander, C, *The Timeless Way of Building*. OUP 1979

Astrachan, O, Berry, G, Cox, Landon, Mitchener, G *Design Patterns: An Essential Component of CS Curricula*. In The Proceedings of the Twenty-Ninth SIGCSE Technical Symposium on Computer Science Education, pp 153-159 ACM Press, 1998

Baxandall, Michael *Painting & Experience in Fifteenth Century Italy* Oxford University Press 1972.

Bayle, E, Bellamy, R, Casaday, G, Erickson, T, Fincher, S, Grinter, B, Gross, B, Lehder, D, Marmolin, H, Moore, B, Potts, C, Skousen, G, Thomas, J. *Putting it all Together: Towards a Pattern Language for Interaction Design*. In SIGCHI 30 (1), 1998, pp17-24

Booch, G. *Object-Oriented Analysis and Design* Benjamin/Cummings 1994

Brown, W..J., Malveau, R.C., McCormick, H. W., Mowbray, T.J. *Anti Patterns: Refactoring Software, Architectures and Projects in Crisis.* John Wiley 1998

Effective Projectwork in Computer Science http://www.cs.ukc.ac.uk/national/CSDN/html/EPCOS/EPCOS.html

Forster, E.M. *Aspects of the Novel* Edward Arnold, 1927

Gabriel, R, *Patterns of Software: Tales from the Software Community*. OUP 1996

Gamma, E, Helm, R, Johnson, R, Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software* Addison-Wesley 1994

Jacobson, M, Silverstein, M, Winslow, B. *The Good House: Contrast as a Design Tool* The Taunton Press 1990

Lakoff, G, *Women, Fire and Dangerous Things: What Categories Reveal about the Mind* University of Chicago Press, 1987

McLaughlin P, Sharp, H, Manns, M.L., Prieto, M. *A Patterns Based Approach to the Dissemination of Proven Effective Teaching and Learning of Object Technology*. Monitor, Issue 9, Spring 1998

The On-line Computer Science Teaching Centre In Conference Proceedings 6th Annual Conference of the Teaching of Computing and 3rd Annual Conference on Integrating Technology into Computer Science Education, Dublin 1998. And at http://ei.cs.vt.edu/~cstc

Reference for this paper: *Journal of Computers in Mathematics and Science Teaching: Special Issue CS-ED Research,* 18(3):331-348, December 1999.

Pedagogic Patterns: Successes in Teaching Object Technology. http://www-lifia.info.inip.edu.ar/ppp (last update 12th August 1998)

Wildemeersch, D *Paradoxes of Social Learning: Towards a Model for Project-Oriented Group Work*. Presented at Project-Work in University Studies, Roskilde University, August 1997

Reference for this paper: *Journal of Computers in Mathematics and Science Teaching: Special Issue CS-ED Research,* 18(3):331-348, December 1999.