Design Patterns in Learning to Program

by

Ron Porter, *B.A., Graduate Diploma in Computer Science, B.Sc.(Comp.Sc)(Hons)*
School of Informatics and Engineering,
Faculty of Science and Engineering

November 24, 2006

A thesis presented to the
Flinders University of South Australia
in total fulfillment of the requirements for the degree of
Doctor of Philosophy

Adelaide, South Australia, 2006

Table of Contents

List of Figures

List of Tables

Abstract

This thesis argues the case for the use of a pattern language based on the basic features of the programming language used in instruction for the teaching of programming. We believe that the difficulties that novices are known to have encountered with the task of learning to program ever since the inception of computers derive from a basic misfit between the language used to communicate with a computer, the programming language, and the way that humans think. The thrust of the pattern language idea is that patterns are the essential element in understanding how the mind words in that they are the source of that relationship that we call 'meaning'. What an entity or event 'means' to us derives from the effect that it has on us as living biological beings, a relationship that exists in the 'real world', not from any linguistic relationship at the symbolic level. Meaning, as a real world relationship, derives from the patterns of interactions that constitute being. The meaning that an entity has for an individual is more than can be expressed in a formal definition, definitions are matters of agreement, convention, not the pattern of experience that the individual has acquired through living. What is missing for a novice in any skill acquisition process is meaning, the pattern of experience. All that we can give them using a formal linguistic system like a programming language is definitions, not meaning. Pattern language is the way that we think because it exists at that fundamental level of experience as living beings. The patterns of experience become the patterns of thought through recurrence, not through definition. But this takes time, so in presenting new material to a person trying to learn, we have to present it in the form of a pattern language, the "cognitive map" that drives the problem soving process. Creativity is *always* a function of combining ideas, what is really being created is new meaning, not a program, or a house, or a poem, or a sculpture - these things are mere implementations of meaning. Ultimately meaning can derive only from experience, the pattern of life around us, so creativity is the language of experience, pattern language. The mind is the product of experience, creativity its modus operandi.

Certification

I certify that this thesis does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any university; and that to the best of my knowledge and belief it does not contain any material previously published or written by another person except where due reference is made in the text.

As requested under Clause 14 of Appendix D of the *Flinders University Research Higher Degree Student Information Manual* I hereby agree to waive the conditions referred to in Clause 13(b) and (c), and thus

- Flinders University may lend this thesis to other institutions or individuals for the purpose of scholarly research;
- Flinders University may reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Signed   Dated

Ron Porter

Acknowledgements

As with anything else in life, producing a dissertation involves the use of a pattern language, and one of the patterns in that language is to reflect, at the very end of the journey, on those who have been, knowingly or inadvertantly, fellow travellers, to explore, as it were, the patterns of human connections that make you who you are and the thesis what it is.

My original and enduring inspiration is Lel, who is responsible in ways that she can never know, for this journey, and who, to this day, still whispers in my ear whenever I need reminding of some pattern I have forgotten.

In the patterns of everyday life, family and friends make important contributions of which they are mostly unaware, but which are essential in making any journey like this one possible, and a few friends who deserve particular mention are Sok Kiang Lee, Gisela Bogdan, Lochie and Sam Davies, Jayne White for the insights she gave me into educational and curriculum issues, and, latterly Morgan and Jackie Stringer.

This probable lack of awareness of their vital contribution extends also to many of one's colleagues. Two such contributors that deserve special mention are Denise de Vries and Aaron Ceglar whose input is incalculable. Most of the material on the "epistemic cut" was honed in discussion with the latter, who, much to his own surprise, turned out to be the most philosophically aware of my colleagues, save only for my erstwhile room-mate, Scott Vallance, who was also the source of many insights.

A more conscious contribution was made by Tiffany Winn and Lorraine Harker, fellow travellers on the pattern language route. Tiffany, in particular, opened many doors in the interesting discussions we had along the way.

Through one of those doors stepped Jim Coplien, who as well as contributing to this dissertation directly, introduced me, somewhat unconventionally, to Joe Bergin, whose idea it was to attempt to experimentally measure the effectiveness of patterns in programming pedagogy.

Of course, the most vital contribution is made by one's supervisor, and in in my case I chose wisely. Paul Calder was the most constant of my fellow travellers, and although some of the places I wanted to visit were, to say the least of it, a bit 'weird', he was always there for the ride. His guidance was the most significant input into this journey.

Many others contributed through their presence in the structure of academic institutions like this one, and my thanks go to all who provide the means for such explorations of unfamiliar territory.

Ron Porter
February 2003
Adelaide.

Preface - Scope and Outline

The thrust of this dissertation is that learning to program, indeed learning any skill, involves the use of a pattern language either explicitly or implicitly. This is because any skill depends on the conceptual structure in the mind, a cognitive map, as this is the only way that creative potential can be expressed. However, there is actually nothing new in this, as humans we spend our lives from day one building such knowledge structures in our mind in every field of endeavour, a process that can only be based on patterns of experience, the things that are the same for all of us, and the connections between them, in short, a pattern language. The trouble is that we do this in a largely unconscious fashion, such as in the way that we 'pick up' the grammar of our first spoken language. This unconscious development of cognitive structure, however, becomes inefficient in fields of great complexity or where practice is difficult without a reasonably developed cognitive structure already in place. Programming is such a field because it is based on the strict symbolic logic of the machine which is not familiar from the everyday cut and thrust of life experience that drives many other disciplines, and so there is no obvious pre-existing relevant knowledge structure.

In other words, we cannot rely on the 'natural' resonance of experience in the programming field with everyday experience as can be done, to some extent, in other fields. Therefore the aim of this thesis is to demonstrate how an evolving pattern language can provide the sort of resonances that take advantage of the connections that already exist in the mind, missing in instruction based on pure logic. This is not an entirely unique problem, many fields such as mathematics, philosophy, science and so on, indeed reasoning in general, involve elements of logical thinking. So, it is possible to find, in other areas, methods of driving the acquisition of creative skill that illustrate the power of the pattern process that we are advocating. Therefore, we touch on many strands which may appear at first to be only loosely connected. This is true, in some respects, but the factor common to all of them is that they demonstrate, in some way, the fundamental correspondence between evolution as design for life through patterns of experience and learning as design for life through patterns of experience. Just as evolution involves a codification of experience in the strict 'logic' of DNA sequence, so too does programming involve a codification of experience in the strict 'logic' of machine instruction sequence. It follows, therefore, that cognitive development, learning in short, as design for life through patterns of experience, should provide the means to enable people to learn to program.

So what we are attempting to show here is that pattern language, as an explicit rather than implicit factor in instruction, is the way forward in crossing the gap between general thinking based on everyday experience and that required for dealing with a strict symbolic logic. This attempt necessarily involves exploring many areas of knowledge to illustrate our argument, and from this exploration it is clear that the cognitive structure involved in any task must reflect the objective conceptual structure (the pattern language) of the field concerned, and this applies in every human skill. While we concentrate our attention in the programming domain on the imperative-procedural-OO paradigm, we do this for the sake of clarity, not because the pattern process applies only here. Ours is a philosophical task, an exploration of the issues involved in acquiring a skill, not a technical one, so, inevitably, the dissertation involves a conceptual (philosophical), even a narrative, flow rather than a logical (technical) one. That such a non-technical and non-reductionist approach is necessary is shown by the moral core of the pattern language idea, the proposition that order and coherence are properties of the whole, not the parts of which it is made.

In Chapter 1 we establish the scope of the issue dealt with in this dissertation - the use of pattern languages to address the problems that novices exhibit in learning to program - and the thrust of our argument is that it is the difference between how humans think and the logical rigor and mechanical nature of programming languages that lies at the heart of the matter. As Christopher Alexander's pattern language concept is directed at the process of designing solutions to problems in the 'real world', we introduce it as a means of providing a cognitive map for novices in the programming domain.

Chapter 2 provides an overview of prior work in the fields opened up for discussion in the introduction in an attempt to provide a broad philosophical basis for the project. Although the idea of a language of patterns is relatively new, patterns themselves have long been recognised as a significant factor in human thinking processes. We examine their use in terms of the learning process and educational practice, and, more particularly, in introductory programming where we find that ideas such as "chunks" and "schemas" which resonate with the pattern concept predate the explicit use of patterns in programming pedagogy. The main aim of this chapter is to introduce pattern theory in both its classical and Alexandrian forms so that we can set the context for its place in pedagogical theory.

As designing any artefact, including a program, is a creative act, we explore, in Chapter 3, the roots of creativity in everyday human experience. Again we are attempting to establish the problem in its widest context, so the discussion here is mostly philosophical in spirit, leaving the psychological and educational aspects to be dealt with elsewhere.

Chapter 4 argues that the source of the difficulties exhibited by novices stems from the fact that instead of attempting to adjust our pedagogies to better fit how the mind actually works, we have persisted in trying to modify the mind to fit the programming system, to 'make people think like computers', in effect.

A designed artefact of any complexity requires generativity, the combination of multiple concepts to a unity of purpose, and Chapter 5 attempts to demonstrate that the fundamental function of language is generation not communication. Before we can communicate any complex idea, we have to have generated it out of the simpler ideas that constitute its components. Language-as-conceptual-understanding, therefore, is the factor that underlies the human condition, that drives all creativity.

Having explored the role of patterns and of language in human thinking separately, we attempt, in Chapter 6, a synthesis based on the pattern language concept developed by Christopher Alexander. Here we are concerned with the use of pattern languages in education, so our examination is mainly confined to the pedagogical implications of Alexander's theory, which means that there are significant differences with pattern practice as it has developed in software engineering.

In particular, we emphasise the use of pattern language diagrams, an aspect of Alexander's thinking that has not caught on in the software field, and Chapter 7 uses a simple programming exercise as a work-through to demonstrate how the language generates the solution. This is not a logical or mechanical process, the programmer still needs to make creative decisions along the way, but the pattern language diagram clarifies the points at which such decisions are needed and the arrows between the nodes in the network point to the various options available at each of these junctures. As the purpose of our example is to illuminate the actual process involved in using a pattern language to generate the design of a program, we have, of necessity, had to simplify - almost to the point of absurdity in terms of their own domains - all aspects, the patterns, the language, and the programming example itself. But our purpose is not to analyse these aspects, but to illuminate the complex interaction between them. So everything in this chapter is constrained by the necessity for clarity in exposing the *dynamics of process*, an undertaking that we found to be almost impossible in fact.

Because programming is an activity that occurs in the human mind there are psychological implications, and these are the subject of Chapter 8. Here we are concerned with how novices acquire meaning, so our examination is mainly concerned with the problem of *meaning*, what it means for an element of thought to mean something, as this, we believe, is the major implication of Alexander's ideas - it is always meaning, conceptual order, that is being designed, no matter which 'material' domain is involved. This is somewhat of a divergence from the more conventional psychological investigations of programming undertaken elsewhere in the literature, particularly by the Psychology of Programming Interest Group, but our investigation, as always, is coloured by its concentration on the development of what might be termed the 'programming mind.' Most psychological research is driven by empirical concerns that fail to address our fundamentally philosophical approach.

Again, in Chapter 9, our investigation of the psychology of learning to program, we diverge somewhat from more conventional treatments, as we base our analysis on the process of developing expertise in general. The case study of the apparently 'trivial pursuit' of reciting large sequences of digits from memory demonstrates how a pattern language is developed and used to drive expert, even world record, performance of a task.

Chapter 10 outlines the three attempts we made to measure the effectiveness of using pattern languages in teaching people how to program. Although our experiments foundered on the difficulty of balancing the motivation of attendance at 'special' pattern sessions and the need of students to maintain progress in their normal non-pattern programming coursework, we felt that our efforts were of value, both in directing any future attempts, and in illustrating the difficulties involved in measuring performance in any mental activity such as programming.

Any investigation that attempts to cover as much ground as we do in this dissertation, inevitably leaves many loose ends dangling in the breeze. The main task of the conclusion is therefore to gather as many of the threads together as possible, and to demonstrate that the unifying principle is the correspondence between pattern language as a basis for programming-as-design and the functioning of evolution in the derivation of complex natural form, evolution-as-design. Like evolution, programming proceeds on the basis of patterns of experience, not the logical progression of formal symbolic systems such as DNA or programming languages, and it is pattern language rather than logical rigor that drives it.