Patterns: Lust for Glory

Malcolm Wieck

School of Computing
Christchurch Polytechnic
Christchurch, New Zealand
wieckm@cpit.ac.nz

ABSTRACT

Pattern frameworks have emerged as a powerful if not yet pervasive tool for the continuous improvement of software authorship, teaching, business administration and building design. The concept of somehow storing proven solutions to repeating problems in a readily-retrievable form has enormous appeal to professionals in all walks of life. While there have emerged some excellent templates for the creation of effective patterns there are a number of alternatives that each offer something to attract different pattern users. This paper reviews those observed so far, considers their features and attempts to recommend a preferred version or versions in the light of developed criteria.

KEYWORDS

Patterns, frameworks, software authorship.

1. INTRODUCTION

Patterns are not new. Using them to achieve the holy grail of software re-usability is new-ish. They are not, however, "... a silver bullet. They certainly won't single-handedly solve the software crisis." (Rising, 1998, p. 3) Patterns are platform independent, web-deliverable, open-sourced, stage-delivered, extensible objects. They embody named nuggets of insights into recurring problems; within a given context they provide the core of a proven solution that can be implemented by a user to solve any number of given problems. Christopher Alexander said:

"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, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Alexander, 1977).

Following Alexander's pattern language approach, adopted to create lasting architecture of quality, a number of notables in the software industry have grasped an opportunity to impose a practical, proven style of presentation into the quest for better software authorship. At CPIT, The Pattern Language of Teaching project is an example of an application outside software development where patterns are being written.

In order to obtain a pattern to use, users must search for an appropriate pattern from a catalogue and then interpret its use in their own implementation of the given solution. The user will typically apply the solution in different ways from those encountered (and perhaps even intended) by the pattern author, so a thorough understanding of the context and forces acting on the problem is required. If this is not achieved, they might fail to match the required conditions under which the pattern should be applied with the appropriate pattern. To obtain a successful match, the user needs to be sure that they have the right pattern for the problem they need to solve. The presentation of the pattern must display the information contained within, in a transparent, logical, easy-to-follow way. It is thus held that the pattern format is an important factor in the ultimate usability of any pattern, or complete pattern language and that care should be taken in establishing the chosen format if the hard-won knowledge is not to be hidden or made difficult to apply.

It is further held that a concisely structured format, with contrasting headings separating the sections, offers advantages for the hard-pressed developer over more verbose, story-like versions that require more time to read through. A small-scale, pilot experiment was carried out using two different versions of the same patterns from an organisational pattern language. Organisational patterns were chosen since the same patterns were readily available in both formats. A classroom tutorial of a stage three information systems paper that teaches the use of software patterns was used as a vehicle for the experiment that looked for indications of greater suitability of one or the other format. The results are given below.

2. HORSES FOR COURSES

There are several available formats for writing out patterns, but the whole point of all patterns is the same: to help the user by communicating knowledge. For this knowledge to be of any use to a potential user it must be retrievable; it must also be effectively applied to an appropriate problem. Readers must therefore search for solutions that approximately match their problems before they apply the solution. Exactly where and how they search is outside the scope of this paper and is something the author will address in the future.

John Vlissides, an editor of PLoP2 (PLoP 2, 1996) had this to say about pattern forms following his workshop on software architecture in Dagstuhl, Germany where he outlined habits all pattern writer should try and acquire:

The pattern concept is new enough and the subject matter complicated enough that some people have a hard time seeing the point of it all. Everything that can be done to make a pattern approachable should be done (Vlissides, 2001).

Despite the interest in buildings and communities spawning from Alexander's work, it is reasonable to assume that software authors will look for patterns to inform their work in a catalogue that bears the name of some aspect of computing, rather than building architecture. Although that is logical behaviour, it is perhaps unfortunate that it will at the same time deny access to those patterns that have been abstracted to illuminate higher-order thinking. Still, the commercial imperative rules many people's lives, so this will be a price many will gladly pay. While some really neat pattern ideas, sourced from alternative catalogues may be missed, the current problem will get the attention it quickly needs, in the form of a workable solution.

Few followers of the creed would be surprised to learn of the existence of a pattern for writing patterns themselves; Gerard Meszaros and Jim Doble published such a pattern language in the (Meszaros and Doble, 1998) The existence of a pattern for writing patterns might lead the first-time enquirer to believe that since the patterns were to be written by disciples of the faith, surely all patterns would have the same format? Yet pattern authors have used a number of different pattern formats in which to hold their valuable knowledge; that the existence of a solution in no way guarantees its wholesale implementation. Apparently then, groups of authors have clustered into areas of common interest and used the pattern formats they feel are best suited to hold their collective wisdom or else formats with which they are most comfortable.

Another word from Vlissides from the same workshop referred to above, confirms the need to construct patterns that are both easy to read and follow effective formats:

Patterns have the insidious property that they are usually perfectly understandable to people who are familiar with the problem involved and its solution. Such people have used the pattern before unconsciously. Thus when they see the pattern, they can recognize it immediately, even if it isn't presented very well. The real challenge is to make the pattern understandable to people who have never run across the problem before (A summary of the workshop appears in ACM Software Engineering Notes, July '95, Vol. 20, No. 3, pp. 63-83).

It is held that for a given scenario and pattern language, the choice of pattern format will have an effect on the both the speed with which a required pattern can be found and also the accuracy with which the pattern can be matched to a given problem scenario. A simple experiment was used to test this hypothesis. Subjects were presented with a problem requiring the retrieval of useful patterns from a collection of useful and not so useful patterns. One half of the subjects were presented with a pattern in Alexandrian Form; a fairly lengthy, verbose format. The other half of the subjects was given a more concise format, due to the work of James Coplien and the Gang of Four (GOF) (Coplien, 1995). The patterns were graded in terms of their appropriacy and the ability of the two groups to select the three most appropriate patterns was then measured.

The two formats used for the experiment are given in more detail below.

3. ALEXANDRIAN FORM

This format is taken from Alexander (1977, p. x), since the patterns represent things that will be built in three dimensions and will be viewed by others; a picture precedes all other parts of the pattern. The section headings and short descriptions follow. Numbering has been added to aid reading.

1 Picture - showing the archetypal example of that pattern.
2 Introductory paragraph - to set the pattern's context.
3 Three diamonds ? ? ? - to mark the beginning of the problem.
4 A headline - in bold type, giving the essence of the problem in one or two sentences.
5 The body of the problem - empirical background, evidence for its validity.
6 The solution - in bold type, the heart of the pattern describing the relationships required to solve the problem in its context, stated in the form of an instruction.
7 A diagram - showing the solution in diagram form.
8 Three diamonds ? ? ? - to mark the end of the main body of the pattern is finished.
9 Finally, a paragraph is written that ties the pattern to all the smaller patterns in the language that are needed to complete this pattern.

Alexander claims two main purposes behind his format: "First, to present each pattern connected to other patterns, so that you grasp ... all 253 patterns as a ... language. Second, to present the problem and solution of each pattern in such a way that you can judge it for yourself, and modify it, without losing the essence that is central to it." (1977, p. xi) This was chosen to represent the fuller pattern format standard, partly out of deference to the pioneer author and partly because in terms of verbosity it is nearer one end of the spectrum than most. For the purposes of the experiment a reasonably stark contrast was needed to highlight any differences in pattern usability due to this factor.

4. COPLIEN FORM

The translation of the organisational patterns chosen from the modified GOF format has retained the essence of the original but omits the picture.

The section headings are:

1 Pattern Name
2 Problem description
3 Context
4 Forces or tradeoffs
5 Solution
6 Example
7 Resulting Context
8 Design Rationale.

James Coplien's work on organisational patterns "... attempts to use patterns in a generative way." In this way, patterns perform more than simple single problem solving duties; used comprehensively and with sensitivity, they "... should help us build new ones." (Coplien, 1995, p. 184) Coplien appears in sympathy with Alexander's lofty ideals and believes patterns should possess an almost spiritual quality - referred to as a "quality without a name." (QWAN)

5. METHOD

The subjects were given extracts from pattern catalogues in one of the two pattern formats described above and an organisational problem in text form. Extracts comprising six patterns were chosen to minimise the size of the task and hence make the experiment as subject-friendly as possible. Since this was effectively a pilot study, it was felt that a small-scale task would be appropriate and perhaps inform further studies in this area.

Space limitations preclude the presentations of the catalogues and question paper, but suffice to say the problem was a piece of pure text, 103 words long and it outlined an organisational weakness. Four patterns represented reasonable solutions to the problems the organisation faced, with one in particular - pattern 'C'- judged to be especially effective. Subjects had to select and rank the three most appropriate patterns given only 5 minutes in which to read the question and peruse the pattern catalogues for their solutions.

The experiment was conducted first on a group of final year BBComp students who had attended a lecture on the use of patterns and subsequently on members of staff in the School of Computing. See Fig 1. Of the staff sample, only one was known to have seen the lecture, but as practicing business computing professionals, they could be expected to be familiar with the principles of the material. See Fig 2. Two patterns were included as distracters (patterns 'A' and 'F'); their selection would indicate the subject had failed to appropriately match the problem to the available patterns. The names of the six patterns chosen for the experiment were:

A. Size the Organization D. Form Follows Function

B. Developer Controls Process E. Architect Also Implements

C. Code Ownership F. Apprentice

Two indicators of success were employed: a pattern search was deemed successful if 'C' was ranked first or second choice. A second indicator of success was the absence of patterns 'A' and 'F' in the rankings, they having been deemed to contribute little or nothing to the problem under consideration.

6. RESULTS

Results as in Tables 1 and 2.

7. SUMMARY

The figures obtained from this pilot study are not conclusive; the sample size is small and the variations in the two groups are thus not very significant. The initial indication of the first success criterion is the reverse of that expected for the student sample and in accordance with predictions for staff. The respective concise and verbose format percentages were 67% and 100%, indicating students were able to use the verbose format more efficiently than the concise format, although a higher percentage of concise format patterns placed pattern 'C' first (67% against 56%). For staff the figures were 63% against 50%, again, not conclusive. (The temptation to conclude that members of staff are more predictable than students will be resisted.) The second success criterion was even less conclusive, with identical scores of 93% for students indicating zero difference in format choice and 96% and 92% for the two formats for staff.

8. FUTURE PATTERN-WRITING DEVELOPMENT

The following are some observations and minor concerns over the way pattern writing is developing. First, the kind of language used in naming and subsequently describing the pattern is sometimes overly jargon-filled, idiomatic and occasionally downright quirky. For example, the applicability of the following patterns is difficult to infer from their chosen names: the reader's personal interpretation: "Smoke filled room", "Moderate Truck Number" and "Buffalo Mountain." In contrast, some are much more easily understood: "Three To Seven Helpers Per Role", "Developer Controls Process" or "Let the Tools do the Work" would probably be more easily interpreted and hence selected or discarded with minimum investment by the pattern searcher.

Coplien/GOF (Shorter Format) Ranking			Alexandrian (Verbose Format) Ranking		
1	**2**	**3**	**1**	**2**	**3**
C	B	E	C	E	B
B	D	C	C	E	A
C	D	B	B	C	D
C	E	D	D	C	B
C	D	-	D	C	B
C	D	E	C	D	B
A	B	D	C	D	E
C	B	D	D	C	B
D	E	F	C	B	A

18 students in all, 9 students given the shorter format, 9 the longer format

No. of ‘C’ Answers ranked 1 or 2 = **6** No. of ‘C’ Answers ranked 1 or 2= **9**

‘ A’ or ‘F’ discarded = 25/27 = **93%** ‘A’ or ‘F’ discarded = 25/27 = **93%**

% of ‘C’ answers ranked 1 or 2 = **67%** % of ‘C’ answers ranked 1 or 2 = **100%**

Figure 1

Responses from Students, having had “patterns” lectures

Coplien/GOF (Shorter Format) Ranking			Alexandrian (Verbose Format) Ranking		
1	**2**	**3**	**1**	**2**	**3**
E	D	C	C	D	E
C	E	B	D	A	C
B	C	D	B	A	E
E	C	A	D	C	E
C	E	B	E	D	B
D	B	E	C	B	D
C	D	B	D	E	C
E	D	C	C	B	D

16 Staff in all, 8 staff given the shorter format, 8 the longer format.

No. of ‘C’ Answers ranked 1 or 2 = **5** No. of ‘C’ Answers ranked 1 or 2 = **4**

‘A’ or ‘F’ discarded = 23/24 = **96%** ‘A’ or ‘F’ discarded = 22/24 = **92%**

% of ‘C’ answers ranked 1 or 2 = **63%** % of ‘C’ answers ranked 1 or 2 = **50%**

Totals for both staff & students: **11/17 = 65% (shorter) 13/17= 76% (verbose)**

Ideally ‘A’ or ‘F’ discarded and ‘C’ answers ranked 1 or 2 would each be **100%.**

Figure 2

Responses from Computing Staff, having had no “patterns” lectures

If the meaning of the pattern names is rather obscure to all but those involved in the pattern's creation, the chances of the patterns being quickly found by a user outside the circle must be diminished. While numbers of software patterns are still small in proportion to total code written, this obscurity issue is not likely to be a major problem. However as pattern numbers grow and inexperienced pattern users also find themselves trawling the books and web sites for solutions, the need to make the patterns easier to find and easier to understand and apply increases. Authors should perhaps have an eye to the future use of their work and allow for different cultural backgrounds, inexperience and first language. Certainly, a group of American software developers are entitled to call their patterns whatever they wish - it's their knowledge that builds the pattern after all. The penalty however is that potentially obscure naming may serve to exclude many potential pattern users, a situation surely contrary to the spirit of knowledge sharing found in practically all fields investigated during this paper.

Will pattern writing stop, or reach saturation point? In time, will there be so many patterns available for builders of software that the need for creativity evaporates? Vast libraries of solutions for every conceivable organisation's project requirements, covering analysis, design, implementation and maintenance phases will perhaps one day be available. Remembering that the better patterns currently help us write new ones and assuming artificial intelligence (AI) continues to mature, it is not too far-fetched to suppose that soon, the AI-capable pattern generator will become a reality. Voice actuated software production might simply become a matter of the user discussing business requirements with the AI front end of the pattern-writing software, which then searches for available patterns to compose the software before offering a range of solutions. The user could then select from a range, ranked on, say, Cost Benefit Analysis.

Standard practices of staged delivery, prototyping, receiving user feedback could be retained. Users would get what they asked for - no more and no less and they would be confident that they were obtaining proven code with high quality, tried and tested documentation to support it. Corrective and perfective maintenance would be greatly simplified, anticipated even, with known, working enhancements priced in advance and available on demand.

Crystal ball gazing aside, it is possible that the patterns writing fraternity may fracture with the dividing line being either the method of creation or treatment of the patterns once they are written. Viewed as a short-term solution to an immediate problem they can provide a quick fix to those that seek it. Such authors in this camp create so-called non-generative patterns. For those with higher intentions, perhaps carrying the spirit of Alexander and others who follow his work, the aim is to build generative patterns that "... are concerned with the act of building. They are meant to embody the Alexandrian ideals." (Evitts, 2000 p. 189) Evitts refers to a pattern's connections thus: "... within a pattern, establish its value and generativity by showing how it connects and combines the elements of the solution in a way that provides opportunities for creative use." (2000, p. 195)

9. CONCLUSION

The belief is still held that there are pattern formats that any given audience will find easier and more effective to use than others. Anecdotal evidence from colleagues supports this. The pilot study conducted neither supports nor refutes it. The Pattern Language of Teaching project underway at CPIT will need to adopt a format that does justice to the level of effort that will be needed to create it. The format should allow access to the knowledge from as wide an audience as possible not just the narrow group of authors "in the know", after all it is those without the knowledge that need it most. As Robert Martin said: "Cataloging (sic) [patterns] makes them accessible to those of us who have not stumbled across the techniques on our own." (Martin, in Rising, 1998)

ACKNOWLEDGMENTS

I would like to thank the students of the final year Bachelor of Business Computing course for their cooperation. I would also like to thank Dr Mike Lance of CPIT, for his inspiration and continued support.

REFERENCES

Alexander, C., Ishikawa, S., Silverstein, M., (1977)
"A Pattern Language: Towns - Buildings - Con-

struction." Oxford University Press, New York.

Coplien, J.O. (1995) "A Generative Development-Process Pattern Language, Ch 13 in Coplien, J.O. and Schmidt, D.C. (Eds.) "Pattern Languages of Program Design." Addison-Wesley.

Evitts, P., (2000) "A UML Pattern Language." Macmillan Technical Publishing.

Meszaros, G., and Doble, J. (1998) "A Pattern Language for Pattern writing." in Martin, R. et al (1998) "Pattern Languages of Program Design." Addison-Wesley.

Martin, R. (1998) "PLoP, PLoP, Fizz, Fizz." Article in Rising, L. (1998) "The Patterns Handbook." Cambridge University Press.

Rising, L. (1998) "The Patterns Handbook." Cambridge University Press.

Vlissides, J.M., Coplien, J.O., Kerth, N.L. (Eds) (1996) "Pattern Languages of Program Design 2." Addison-Wesley.

Vlissides, J.M., (2001) "Pattern Hatching - Seven Habits of Successful Pattern Writers." Retrieved 13 May 2001 from the World Wide Web: http://hillside.net/patterns/papers/7habits.html.