



**Peter F Brown**  
**Independent Consultant**  
Transforming our Relationships with Information Technologies

# Introducing Pattern Languages

Some comments and ideas for the development of a  
Pattern Language for guiding governments in service  
transformation

March 2011

Peter F Brown

Entire contents © Peter F Brown, 2011. All Rights reserved

For further information please contact me at [consulting@peterfbrown.com](mailto:consulting@peterfbrown.com)

Peter F Brown  
Independent Consultant  
P.O Box 49719  
Los Angeles, CA 90049-0719  
USA

## Purpose

This note has been prepared to explain “Pattern Languages” and how the paradigm might be applied to create a tractable standard for use by governments, consultants and developers in helping the process of transforming the delivery of public services. This is one of the stated objectives of the OASIS Transformational Government Framework<sup>1</sup>.

The idea of Pattern Languages, as a process for analysing recurrent problems and a mechanism for capturing those problems and archetypal solutions, was first outlined by architect Christopher Alexander<sup>2</sup>.

The value of a Pattern Language is that remains readable and engaging whilst providing basic hooks for further machine processing. I will return to this further stage in a later note.

## Background

The OASIS Transformational Government Framework (TGF) technical committee (TC) has recently released a draft ‘TGF Primer’, intended to introduce the main concepts covered in its eponymous Framework.

The concern of the TC is to render the Framework – currently structured as a policy paper - in a tractable form. The challenge has been to find a form that retains the essential prose and urgency of the issues and approaches raised and that is readable by both senior management and policy makers, as well as by technology analysts, designers and developers: everything from ‘Concept Maps’ to full-blown OWL have been proposed as possible solutions.

I recently raised the prospect of developing the TGF as a ‘Pattern Language’ and this document explores the rationale for this. The ideas raised here could in fact be applied to any number of business strategy areas.

---

<sup>1</sup> See [http://www.oasis-open.org/committees/tc\\_home.php?wg\\_abbrev=tgf](http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tgf)

<sup>2</sup> See Christopher Alexander *Notes on the Synthesis of Form, A Pattern Language* and *The Timeless Way of Building*

## The Problem

In a word – communication. Subject matter experts of any domain will recognise similar problems occurring time and again as well as certain ‘winning solutions’ that seem to address these problems time and again. Frustration often arises when these experts recognise that others are facing similar problems but that they (the domain experts) have no satisfactory means by which to communicate their solutions to those others. The experts would like to share but lack a clear form and method by which to do it.

Nowhere is this more noticeable than in the consultancy business working with multi-disciplinary domains or in large-scale organisations that need everyone from senior management to junior developers and front-line staff to share a common strategy and vision. Nowhere is this more acute than in such organisations that are undergoing profound change or transformation.

Although there may be an implicit or explicit organisational change strategy, this will be expressed and realised in many different ways, from top-level Board, Ministerial or Departmental policy statements, all the way through to specific technology infrastructure and service changes.

In a program of IT-driven transformation (and the TGF covers such scenarios for the public sector delivery of services), the problem arises when trying to find a common way in which to express what is needed, that is clear and understandable to both managers and IT developers. Managers are used to policy documents; developers to clear algorithms, formal models and executable code.

Meszaros and Doble highlight the main issues that need to be addressed<sup>3</sup>:

- Keeping the solution to yourself doesn't require any effort
- Sharing the solution verbally helps a few others but won't make a big impact in your field.
- Writing down your understanding of the solution is hard work and requires much reflection on how you solve the problem.
- Transforming your specific solution into a more widely applicable solution is difficult.
- People are unlikely to use a solution if you don't explain the reasons for using it.
- Writing down the solution may compromise your competitive advantage (either personal or corporate.)

The initial work already undertaken in the TGF Primer has gone a long way to bridge this communications gap already and much of the work represents a distillation of many experiences into a more widely-applicable framework. What remains to be done is to cast it in a tractable, standards-compliant form. This is where a Pattern Language proves useful.

---

<sup>3</sup> Gerard Meszaros and Jim Doble *A Pattern Language for Pattern Writing*

## The Solution

Pattern Languages provide a means to bridge that communications gap and should help to address issues such as wasted effort from duplicating the same solution from the same type of problem; the lack of a common terminology and definitions of key terms; the inability to extrapolate from specific problems to generic ones; and addressing the broader concern of capturing complex problems with greater insight to the ‘black box’ that often exists between broadly stated initial needs and the delivered solution.

A Pattern Language is not an ‘out-of-the-box’ solution but rather some ‘familiar’ patterns with which a team can work. Quoting slightly out of context (the author is talking about interface design but is equally inspired by pattern languages),

*“Familiar” doesn’t necessarily mean that everything about a given application is identical to some genre-defining product... People are smarter than that. As long as the parts are recognizable enough, and the relationships among the parts are clear, then people can apply their previous knowledge to a novel interface and figure it out... That’s where patterns come in...<sup>4</sup>*

**The Pattern Language provides the basic vocabulary, grammar, structure and plot – but each project has to write its own story.**

Whenever we break any complex problem down, we arrive at some point with a collection of inter-related smaller problems. Each of these can be captured as a distinct *pattern*. Each *pattern* consists of: a problem arising; in a particular context; and a solution to address it.

“Patterns are not solutions. They generate solutions”<sup>5</sup>. Thus is a pattern an abstract representation of some problem, context, and solution that recurs in different contexts. It provides a means for identifying whether a given solution is appropriate to solving a particular problem in a given context.

The pattern is therefore both a description of a ‘thing’ to be achieved and a rule stating how to achieve it. It offers a conjunction of theory and practice and any pattern must be “*Useful, Usable and actually Used*”<sup>6</sup>:

*A pattern must be useful because this shows how having the pattern in our minds may be transformed into an instance of the pattern in the real world, as something that adds value to our lives as developers and practitioners. A pattern must also be useable because this shows how a pattern described in literary form may be transformed into a pattern that we have in our minds. And a pattern must be used because this is how patterns that exist in the real world first became documented as patterns in literary form.*

---

<sup>4</sup> Jenifer Tidwell *Designing Interfaces*

<sup>5</sup> See *Patterns as a Means for Intelligent Software Engineering*

<sup>6</sup> See Brad Appleton *Patterns and Software: Essential Concepts and Terminology*

However, the scope of the issues that we want to deal with often requires a *Pattern Language* rather than just a *pattern* because a single pattern is unlikely to suffice. A solution based on using a single, large pattern (a or 'Large Lump' approach, as Alexander calls it) may not be appropriate in many situations and difficult to re-use – hence the value of a language that brings together a collection of patterns but allows each pattern, or groups thereof, to be re-used in many different situations.

Taking Alexander's principles, a pattern language can be applied in any situation where the following all hold true:

- **Organic Order** – when a 'master plan' simply cannot address the problem, such as when it is not possible to 'fix today what an environment should be like in 20 years', then it is more important to adopt a **process** rather than a plan. Order is created and sustained not from a fixed plan or map, but from using a pattern language and a properly administered process.
- **Participation** is important if not key. 'Resultant chaos' is often cited as an argument against too much participation (and a justification for a 'master plan') but this is not necessarily true, particularly if participation is based on shared principles and commitment to use the process and constructs of an agreed pattern language.
- **Piecemeal Growth** is favoured over 'big bang' solutions. Piecemeal *growth* does not mean a piecemeal *process*. Piecemeal growth and repair of a system or environment imply a gradual sequence of changes and that nothing is ever really 'finished'. Properly executed it means that there are earlier and more frequent deliveries of results in contrast to the 'big bang' approach of knock-down and replacement.
- **Diagnosis and Repair** – the initial form or program arises from the same process of diagnosis and repair that keeps a system stable once mature.
- **Coordination** – all new projects or aspects of an ongoing program of work must clearly explain their relationship with currently adopted patterns and diagnoses. The conformity or not of a new idea with existing patterns will be a priority consideration and continual gap analysis will drive the creation of any new pattern.

All of these would seem to be valid considerations for the Transformational Government Framework. There are clearly **functional similarities** between Alexander's pattern language for architectural design and the need, in government transformation programs, for a high-level yet tractable expression of problems, contexts and archetypal solutions. In recognition that 'nothing is ever finished', particularly in the realm of public sector policy, the **organic approach** is also a realistic alternative to the catalogue of 'big bang' project failures and shortcomings in public sector delivery of services.

## Structure of Pattern Languages

Although there are no hard and fast rules about how to create a Pattern Language, it is a design process based on three concepts, outlined by Alexander in typically poetic fashion: a 'Quality' that is achieved by building 'a living pattern language as a Gate' and through which one passes in order to pursue the 'timeless Way'. The 'Quality' is what we are looking to achieve; The 'Gate' is the pattern language that allows that result to be achieved; The 'Way' is an organic process that allows progressive growth and a particular design or architecture to flourish.

5

A Pattern Language is more than just a collection of Patterns – the individual patterns are not isolated, even if they are independent and can be used independently to handle a specific problem. Rather, all patterns are related as a network of connections, with a loose hierarchy of patterns where each pattern depends on both smaller, more detailed patterns and larger ones within which each is contained. The hierarchy allows one to move easily from the more general to the more specific and this provides clarity in dealing with any specific problem at hand – while the patterns 'above' and 'below' are visible, and this is useful to understand context, using one pattern means focussing on the problem in hand: details lie 'further down'.

Alexander defines a 'good language' as one that is both **morphologically** complete – in other words it is possible to visualise what the patterns together can create – and **functionally** complete – in that the language as applied allows all the 'inner forces' (the problems and contradictions that individual patterns aim to address) to resolve themselves.

Alexander is also clear that good pattern languages are organic: new patterns are added according to context and different collections of patterns can be established for different cultures and polities. This will be particularly important in the field of transformational government: there can be no 'one-size-fits-all' approach and due attention will need to be paid to public policy in different countries and regions of the world.

The software world is peppered with pattern languages developed to address different problems, from the earliest (a set of design rules for Smalltalk applications), through the Gang of Four's<sup>7</sup> work on 'Design Patterns' that offered a practical approach to sharing design problems; Apple's Human Interface Guidelines for the Macintosh, back in 1992; or Tidwell's pattern language for interaction design<sup>8</sup> - all of which quote Alexander's *A Pattern Language* as a key source of inspiration. More business-focussed pattern languages exist also, such as the "Business Strategy Patterns for the Innovative Company"<sup>9</sup>

---

<sup>7</sup> Erich Gamma, Richard Helm, Ralph Johnson and John Vlissides

<sup>8</sup> Jenifer Tidwell *Interaction Design Patterns*

<sup>9</sup> Allan Kelly *Business Strategy Patterns for the Innovative Company*

## Structure of a Pattern

Chris Alexander states that each and every pattern “must be formulated in the form of a rule which establishes a relationship between a context, a system of forces which arises in that context, and a configuration, which allows these forces to resolve themselves in that context.”<sup>10</sup>

6

Applied and understood in a wider sense than the original architectural design problems that Alexander was addressing, this could be roughly translated as a three-part rule:

- The **context** in which a particular problem arises (the *ex-ante* condition) and in which the pattern is intended to be used;
- The ‘system of forces’ or **problem to be solved and** that includes the drivers, constraints and concerns that the pattern is intended to address – Alexander highlighted that this ‘system’ often involved conflicting forces (for example, an architect’s desire confronted with a material limitation) that the pattern should seek to resolve;
- The ‘configuration’ or **solution**.

The precise configuration of a pattern will vary from one pattern language to another, but most have the following structural elements in common:

<b>Name</b>	Not just a label but something meaningful and that captures the essence of the pattern. Giving it a name also makes it easily, and verbally, shareable and citable
<b>Introduction</b>	This sets a <b>context</b> for the pattern and should indicate how the pattern contributes to completing larger (‘higher’) patterns
<b>Headline</b>	A single sentence that captures the essence of the <b>problem</b>
<b>Body</b>	The detailed substance of the problem, the forces and constraints, the empirical background and evidence for the validity of the pattern and indications of the range of possible applications
<b>Solution</b>	Always stated as an instruction, as what ‘needs to be done’, this will describe both the static relationships and dynamic rules that help realise the desired outcome
<b>Completion</b>	This text links the current pattern to related, smaller and more detailed patterns that complete, embellish or otherwise extend the current pattern

Only a ‘Proof of Concept’ or similar test will determine whether these elements are suitable and suffice for any problem area under consideration but these elements are the most commonly recommended.

---

<sup>10</sup> Alexander, *The Timeless Way of Building*

## Methodology

I have evolved my own methodology over time, rooted in the 'Alexandrian method' and in the experiences and feedback of other practitioners. Some key steps are summarised here.

- A first pass of the problem area should reveal the specific features (problems, contexts, solutions) that are worth abstracting as distinct patterns. List them and name them.
- Creating a simple 'concept map' will show how the patterns are related to each other, including hierarchically. The concept map should be extended or pruned as the work proceeds, keeping track of dependencies and cross-references.
- For each of the patterns identified, a short abstract or 'thumbnail' will sum up its essence.
- An agreed structure for each pattern, whether the elements listed above or others, will ensure consistency, readability and even processability.
- Deciding which elements are mandatory and which are optional will ensure that supplementary value can be added when mandatory elements do not suffice.
- Keeping an open mind throughout the process of constructing and completing any pattern will help determine whether or not the pattern is fit for purpose and whether other possible patterns have been identified along the way.

7

This approach will be used to create a 'Proof of Concept' for the OASIS Transformational Government Framework.

## Bibliography

- 8
- Alexander, Christopher, *Notes on the Synthesis of Form*, Harvard, 1964
- Alexander, Christopher, *A Pattern Language*, OUP, 1977
- Alexander, Christopher, *The Timeless Way of Building*, OUP, 1979
- Appleton, Brad, *Patterns and Software: Essential Concepts and Terminology*, 2000, see:  
<http://www.cmcrossroads.com/bradapp/docs/patterns-intro.html>
- Beck, Kent, and Cunningham, Ward, *Using Pattern Languages for Object-Oriented Programs*, Apple Computer Inc, and Textronix Inc, 1987
- Borchers, Jan O., *Pattern Languages in Human-Computer Interaction*, 1999
- Coplien, James O., *Software Patterns*, Bell Laboratories, The Hillside Group, 1996
- Denning, Stephen, *The Springboard*, 2001
- Duego, D, et al, *Patterns as a Means for Intelligent Software Engineering*, IC-AI 1999
- Gamma, Erich, et al, *Design Patterns – Elements of Reusable Object-Oriented Software*, 1995, Addison Wesley
- Gabriele, Richard, *Patterns of Software*, 1996
- Kelly, Allan, *Business Strategy Patterns for the Innovative Company*, 2005, see:  
<http://www.allankelly.net/static/patterns/CorpImaginationV2004.pdf>
- Meszaros, Gerard and Doble, Jim, *A Pattern Language for Pattern Writing*, 1996, see:  
<http://hillside.net/index.php/a-pattern-language-for-pattern-writing>
- Rawsthorne, Daniel A. *A Pattern Language for Requirements Analysis*, 1996
- Tidwell, J., *Interaction Design Patterns*, PloP'98, <http://jerry.cs.uiuc.edu/~plop/plop98>
- Tidwell, Jenifer, *Designing Interfaces*, 2005, O'Reilly Media