New Design Ideas
Vol.3, No.1, 2019, pp.44-52

ON PATTERN LANGUAGES, DESIGN PATTERNS AND EVOLUTION

Bruno Postle*

Independent researcher, Sheffield, UK

Abstract. Pattern Languages for the built environment and Design Patterns for computer software are closely related fields. This article discusses differences between the two considering the different problem spaces of each. We identify possible reasons for the general acceptance of Design Patterns in computer science and the corresponding lack of acceptance by architects. A novel approach to the design of buildings is proposed using a synthesis of Evolutionary Computing with a Pattern Language as fitness criteria.

Keywords: *Pattern languages, design patterns, evolutionary computing, computer science, architecture, urban design.*

Corresponding Author: *Bruno Postle, 75 Crookes Road, Sheffield S10 5BD, UK, Tel.: +447981460853 e-mail: bruno@postle.net*

Received: *21 December 2018;* ***Accepted:*** *27 February 2019;* ***Published:*** *28 June 2019.*

1. Introduction

This article provides context for a larger ongoing Free Software project for the design of domestic buildings using simulated evolution (Postle, 2013). This software is unique in that the process is driven entirely by what are called Pattern Languages – with the aim that the buildings should meet human needs at every scale – the resulting designs bear a strong resemblance to vernacular or 'traditional' architecture.

This background is interesting in itself, so we will introduce some ideas regarding the built environment that are generally considered outside the architectural mainstream, but which have long had profound influence in the computer science field.

We will explore how we can learn from this existing cross-pollination between the architecture and computer science fields, with the aim to use this information to close the circle and make something genuinely *useful* for the built environment.

The article starts with architectural Pattern Languages; and the somewhat more successful offspring called software Design Patterns; finally we introduce Evolutionary Computation, which turns out to be an excellent match for Pattern Languages, with a Pattern Language taking the unexpected role of *fitness criteria*.

2. A Pattern Language

From the perspective of this non-architect, one of the more interesting things to happen in architecture during the late twentieth century was the discovery of what are named 'Pattern Languages'; originally described by Christopher Alexander, Sara Ishikawa and Murray Silverstein in the classic *A Pattern Language* (Alexander *et al.*, 1977).

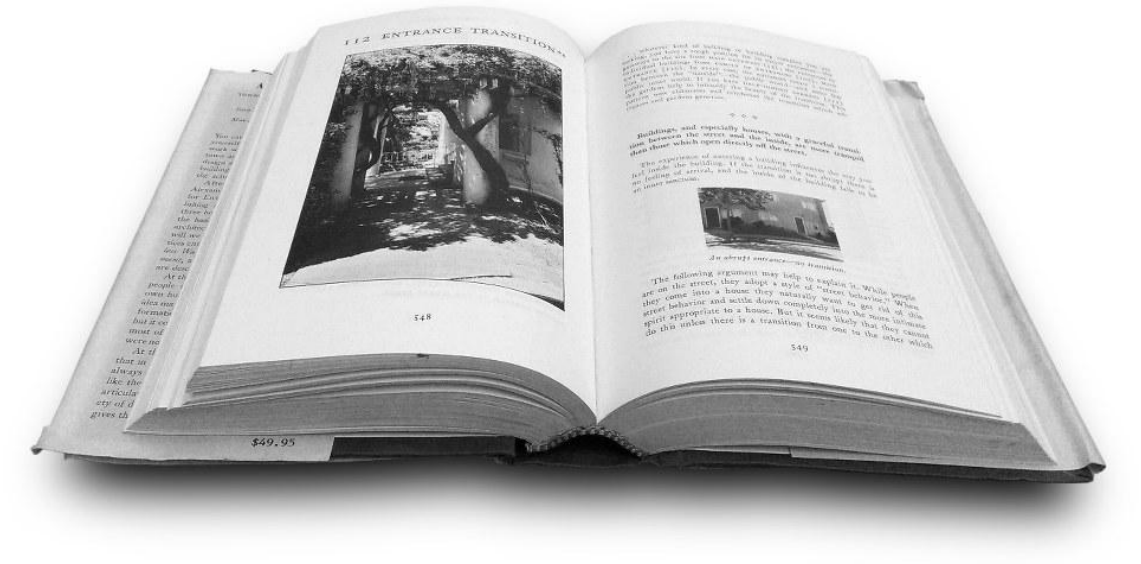

Figure 1. A Pattern Language, 112 ENTRANCE TRANSITION

The ‘patterns’ in this book are not fabric or wallpaper designs, but solutions to typical everyday architectural problems. They can be thought of as each being a guide to meeting a specific human need through architecture. The book consists of a series of 253 chapters in a kind of early hypertext format; the idea is that a collection of ‘patterns’ like this forms a thing called a ‘Pattern Language’ (Leitner, 2015). Each pattern consists of a statement about an aspect of the built environment, noticing problems and suggesting solutions; the pattern is then linked to related patterns that should also be considered at the same time. The patterns are ordered by geometrical scale, with city level patterns such as *9 SCATTERED WORK* at the beginning of the book and *249 ORNAMENT* at the back.

An example is needed, here is a typical pattern from *A Pattern Language* (note that ‘south’ can be exchanged for ‘north’ depending on your location):

> 105 SOUTH FACING OUTDOORS **
>
> People use open space if it is sunny, and do not use it if it isn’t, in all but desert climates.
>
> [...] Always place buildings to the north of the outdoor spaces that go with them, and keep the outdoor spaces to the south. Never leave a deep band of shade between the building and the sunny part of the outdoors.

Crucially, patterns are unlike most of the rest of ‘architectural theory’ in that they are each testable and falsifiable, this is an important point: here is a way of doing architecture as science (Mehaffy & Salingaros, 2015). This falsifiability means that we can and should expect that many of the patterns in the original book would be flawed and replaced over time with something better. Plus, there are potentially any number of other, undocumented, patterns that might suit more obscure situations. Each situation you find yourself dealing-with may require a special purpose Pattern Language, one adapted to the task in hand.

Some of the patterns in the book have very universal appeal, such as *159 LIGHT ON TWO SIDES OF EVERY ROOM*, even *251 DIFFERENT CHAIRS* could summarize

the house style of a generic coffee shop chain. An important thing to notice, which we will get to later, is that the patterns are not a design method in themselves, they are rather a guide for assessing a plan of action, assessing potential designs, or assessing actual places and buildings. *A Pattern Language* has been in print continuously for decades, which says something about the attraction of this Pattern Language approach.

If you are somebody who pays attention to these things you might have noticed that actual real architects generally don't make much use of Pattern Languages, or even use Alexander's book. At best it is thought of as an interesting historical curiosity, at worst it is considered too restrictive of the architect's artistic expression.

Specifically, and this the way I see it, there have been two obstacles to the integration of Pattern Languages with the practice of 'architecture':

One has an origin in what we think of as 'creative design'. What a 'designer' does is something like this: they internalise all available information, they think really hard, and they produce a design as a creative act, the better the designer the better the creativity and the better his resulting design. The point is that when architects have tried to integrate Pattern Languages into this creative design framework, the results have been unspectacular; why bother with patterns when all they seem to do is make the process more complex than it would be otherwise? Maybe Pattern Languages are just not a good fit with the way we do creative design?

I should point out that Alexander himself considers A Pattern Language to be incomplete or inadequate, "my biggest failure" (Alexander, 2010). The problem being that the Pattern Language itself was taken to be a series or rules, or instructions, and the result of using it like this for architecture were often crude or badly formed (Mehaffy, 2011). From the late 1980's onwards, Alexander became more interested in the process of morphogenesis in organisms, describing this as a process of step-wise unfolding where each part of a structure and each stage of development embodies what he describes in metaphysical terms as 'wholeness'. This wholeness itself is expressed in fifteen properties or traits that Alexander believes are inherent in all things that have emerged in this way. Alexander concluded that this adaptive unfolding process is as important for the creation of the built environment as it is in nature (Alexander, 2002).

The other problem with integrating Pattern Languages into the design of buildings, is that if you really do accept Alexander's patterns then you quickly come to the conclusion that the prevailing canon of architectural forms: minimalist detail, bold geometrical shapes, industrial repetition and monumental scale; are kind of 'anti-patterns' that, although many people find them creatively satisfying, don't actually meet human needs (Salingaros, 2017). Be careful though, because thinking like this might get you labelled as *traditional*, often seen as the opposite of *modern*. Somehow, perhaps it is just a flaw in our language that means we can't dissociate a name of a thing from the thing itself, something that would otherwise just be a slightly weird passing design fad (Curl, 2018) from a century ago (to be clear, I'm just talking about *architectural modernism* here), has become associated with all that is progressive, and resistance to it has become associated with everything that is backward, traditional, conservative, Luddite.

Those of us who want a humane, human-scale, ecologically-sane, living environment as promised by *A Pattern Language*, but who don't feel very 'traditional' or 'modernist', what then do we do? Alexander implies that we are going to have to change to working *adaptively*, evolving buildings through incremental changes using the patterns as a guide (Alexander, 1979).

3. Design Patterns

So as we have seen, there is an alternative approach to architecture based on the theory of Pattern Languages that is largely neglected, disdained and sidelined. On the other hand, we have a standard architectural creative design method that *does* seem to be a bit hit and miss. If we judge by the quality of the cities around us, it seems we don't have enough creative geniuses, our geniuses are not good enough, or maybe we can be generous and say they are just not given the opportunity to demonstrate their genius.

However if you come from a Computer Science background, you might think that this Pattern Language business sounds very similar to a big thing in Object Oriented programming called *Software Design Patterns* – you would be right because the two are directly related (Beck & Cunningham, 1987).

Design Patterns came to prominence with the 'Gang of Four' book on *Design Patterns* (Gamma *et al.*, 1994); this collected and codified a series of 23 patterns that had long been part of Object Oriented programming practice, though not necessarily identified as such. Alexander's influence is all over this book, not just the title, but the patterns themselves are written using a similar 'name, problem, solution, consequences' structure.

Software Design Patterns describe a very practical way of working, so much so that even twenty years later they are still central to Object Oriented programming; you will often hear programmers, even those who have never read a book, discussing projects in terms of patterns – the point is that the influence of *A Pattern Language* is everywhere. As an example of the extent of this influence; collecting and describing these Design Patterns required some place to collaborate – the first wiki was devised by Ward Cunningham specifically as a repository for these Design Patterns (Leuf & Cunningham, 2001). So it is true to say that everyday tools like Wikipedia have this Pattern Language influence running right through them.

Adaptation, as opposed to *invention*, is crucial to the idea of software Design Patterns; remember that Alexander also suggests working adaptively with Pattern Languages. Adaptation in this context means that you are supposed to take existing solutions and adapt them to the problem at hand – because it is likely that your specific problem has been encountered before. It also implies building software by starting with a small working system, and extending it – rather than implementing a grand design in one go. This *adaptation* idea, of taking many small steps towards a goal, is also the basis for the closely related systems of *agile software development* – building software by making small incremental changes, but with continuous testing all the way – not to be confused with subsequent management fads with similar 'agile' names (Beck *et al*., 2001).

Agile software development understands that initial design specifications usually change during implementation, sometimes dramatically. We are all familiar with finding ourselves with a task that turns out to be something entirely different once we get started, *agile software development* recognizes that this is a normal state of affairs rather than an exception. Another bit of 'agile' wisdom, which at first sight seems illogical, is the idea that when making these incremental changes, it is best to choose the change that makes future changes easier, rather than aiming directly for the goal.

But how can all this real success in the software industry be reapplied to buildings and cities? Architecture is after all where Pattern Languages started. The problem is

huge, the divergence between our existing systems of building and a humane, living environment seems insurmountable.

4. Design Patterns for buildings?

Well constructed software is *universal*, it can be duplicated millions of times and still behave the same in different locations, on different operating systems, unexpected situations, and different hardware; it is *modular*, with consistent interfaces so it can communicate with, or substitute for, other software created by different people. Like a lot of people, I used to think that there could be some equivalent building type, a mass-produced 'pod' that can be quickly moved around the world, with pluggable services and interfaces. This is a very seductive idea; these pods could be swapped and assembled into larger buildings – together millions of pods would create transient cities as and where they are needed – these cities could split and migrate to new locations leaving no trace behind. Taking the software analogy further, you could imagine an Open Source architecture where designs for these modular pods are shared and improved collaboratively, and then rolled-out worldwide using modern manufacturing techniques.

But buildings are *not* the same as software, buildings are *specific* and software is *generic*, every building has to exist in an actual location which is different to a finite number of other locations. For example: the operating system kernel on the computer I'm using to type this comes with driver modules for vast quantities of obscure hardware, most of which I will never use, it does this so that the same kernel will work anywhere without bothering me with tedious configuration. It can do this because, for software, space is cheap and all distances are small. Rarely used kernel modules don't get in the way; they can be kept around because they don't increase the *distance* to the drivers that actually are being used. This is a one-size-fits-all philosophy.

For buildings, not only is space expensive, but distances are critical. Perhaps I want a house like my computer kernel: a big kitchen, lots of bedrooms for guests, somewhere to park the removal van when I arrive, how about a home cinema? a tennis court at the back, maybe two, I don't have a helicopter yet, but a landing pad might come in useful one day. This isn't so absurd, people *really do* have houses like this. But these things are not just expensive, they all take up real actual space; just by existing they increase the travel distances within the house. More than that, in places where every house is built like this you have to travel past an awful lot of other people's stuff to get anywhere. Distances for people tend to be very significant; when space is sparsely occupied, travel distances are further, entailing additional costs and other negative effects (Duany *et al*., 2001).

Software is useful when it saves people from doing repetitive or unnecessary tasks. Efficiency in buildings and the spaces between them is very different to efficiency in software, efficient buildings are packed together, consuming the least amount of space and materials for the most amount of utility; the degree of modularisation is largely irrelevant to this utility, and since all sites are *specific*, modularisation can only ever have detrimental effects on the use of space. By this I mean that a *generic* pod-room or pod-house will always be just a little too big, or a little to small or just the wrong kind of shape. Efficiency in software comes from modularisation and mass duplication, the space that this *generic* duplicated software takes up is largely irrelevant.

Going back to the beginnings of Design Patterns in software, I want to draw attention to something notably different in the original 1987 paper by Kent Beck and Ward Cunningham (Beck & Cunningham, 1987). The Pattern Language presented in this paper is actually for a graphical windowing application, and the patterns describe arranging windows on a screen, very much like arranging bits of building on a plot of land, the inheritance from *A Pattern Language* is clear. But although this was where software Design Patterns started, where they developed and found success was *not* in the field of graphical user interfaces, *none* of the patterns in the 'Gang of Four' book of *Design Patterns* are about positioning widgets on a screen, rather they are about how you go about wiring them together behind the scenes.

Consider that this distinction is the reason why the pattern approach has been so effective in software engineering. With software Design Patterns there is a very direct relationship between the problem and the solution, the solution is simply the right way to do it for any particular situation. If you find yourself with a software problem where two Design Patterns are conflicting, where there is an overlap in functionality which means that you can't implement either pattern in full, then *this is a sign that you are doing something wrong*. But buildings are not like computers, in a building there is a finite amount of space where everything has to happen, trying to implement one pattern from a Pattern Language in this space requires an intervention, say a wall or a door or window, that inevitably interferes with some other pattern that requires something else in that exact same space. The consequence is that when building with patterns there is a constant trade-off, a process of give and take that necessarily results in compromises. This is an important point; every built space *in the real world* is the result of a balance of needs and costs. If we are building with a Pattern Language, then *each pattern can never be implemented in full*.

And for the same reason, successful software Design Patterns are generally not about the graphical user interface; computer screens are a limited space, demanding compromise, much like a building plot.

So, as Free Software people, what can we bring to the design of buildings and cities that is fundamentally different from architecture as we know it?

4. Adaptation

In summary, a book, *A Pattern Language*, sets out a path leading to a built environment that is humane and human-scale.

This Pattern Language approach was devised for creating buildings and places, but has been far more successful in the software industry. The influence in the architecture profession is somewhere near zero. So is there any chance of Pattern Languages ever having the impact on the built environment they were intended to have?

Pattern Languages are a solid foundation, but modularisation: creating libraries of refined reusable designs like we do with software isn't a good solution to the problem. Software Design Patterns are just great, and are basically the same thing as Pattern Languages, but their phenomenal success doesn't necessarily translate to help us create actual buildings in the real world.

Let's go back to the principle of *adaptation* as a design method that is central to both Alexander's practice as an architect (Alexander, 1979) and to 'agile' software development. *Adaptation* here means a very specific thing; *adaptation* is a process of *evolution through incremental change*, think of it as different to *invention*, which is the

application of *inspiration* to create something *new*. Of course the typical process of creation involves a bit of both things, but it is helpful to think about which processes are *adaptive* and which are *inventive*.

Specifically, life and life's evolution through selection is adaptive, *Darwinian natural selection* means that organisms that are the *best fit for the environment* are the organisms that go on to produce the next generation (Darwin, 1859). The environment *forms* living organisms through selection. Selection of living organisms can be manipulated by humans, as has been the case with domesticated animals and plants, here our human interference a just another part of the environment. *Evolution in nature happens through small incremental changes, there is no inspiration involved.*

5. Evolutionary computing

Here is perhaps a solution to the problem. The problem is that Pattern Languages for *architecture* require an incremental, adaptive, approach. But making a decision for any single building situation involves balancing potentially dozens of patterns. Then each of these design decisions can have unseen ramifications beyond the task in hand, how can a single person juggle all these variables at the same time? And making these decisions again, and again, and again, since *what we actually mean by 'design through incremental change' is that there is going to be a lot of repetitive decision making*.

(Of course none of this would be a problem if you design by starting with a grand 'creative concept', fill in all the details later, and don't worry too much about the bits that don't quite fit, but this is an approach to design that leads to all too many failures in our built environment)

In computer software terms, balancing multiple variables like this is an *optimisation* problem, specifically this is the kind of hard problem that can be tackled using *Evolutionary Computation* (Eiben & Smith, 2003). Evolutionary computing closely follows nature's example; it does this not by making changes to a single individual, but by creating a population of individuals and simulating environmental pressures on them. These environmental pressures select for survival the part of the population that is a better fit for the environment. This process is then repeated again and again, the population is repeatedly evaluated for selection by the *fitness criteria*, and the next generation is usually a better fit for the environment.

So how about trying a different approach altogether for designing buildings? Can we create software that designs buildings in an evolutionary way using Pattern Languages as *fitness criteria*? What if we can describe the steps necessary to unfold a building using a genetic 'code', and that genetic code can be mutated or crossed with the 'code' of other buildings? What if we put these 'buildings' in a simulated environment of a Pattern Language, of daylight and surrounding landscape? Would this be enough to begin designing a humane, living environment, as promised by *A Pattern Language*?

Further than this, would it be possible to control the process of building evolution so that the state is 'whole' at each step? This would need to consider Alexander's fifteen properties in addition to the Pattern Language, with the purpose or use of rooms being a secondary consideration. Allowing the function to follow the form like this would run counter to accepted architectural practice, but if we look at buildings that have survived for centuries, the use of individual rooms, and even the purpose of the building, may have changed many times during the lifetime of the structure. Such buildings may not

be a close-fit for purpose, but they may be a close-fit for general human use, the immediate environment, daylight and a 'good' geometrical shape. This would be a suitable topic for further investigation.

6. Conclusion

This article is not the place to describe the work that has been undertaken to implement these ideas, this can be found elsewhere (Postle, 2013; 2018).

Pattern Languages have already proven to have a profound contribution to the computer science field, it remains to be seen whether *A Pattern Language* will have the equivalent influence on architecture and the built environment as was originally intended.

A final quote from Alexander:

> "Why would computer scientists and software engineers suddenly become responsible for the form and structure of the built environment? Is that not the province of architects, planners, agricultural experts, forestry people, and civil engineers? It ought to be. But the members of these professions are not taking responsibility for the generative approach to living structure – and so cannot produce it. And, as far as I can see, they do not see it coming, and are not preparing themselves to take it on, mentally or professionally. Therefore it will fall to someone else to do it instead."
>
> "I want you to realize that the problem of generating living structure is not being handled well by architectural planners or developers or construction people now, and the Earth is suffering because of it. I believe there may be no way that they are ever going to actually be able to do it, because the methods they use are not capable of it. For you it is different. The idea of generative process is natural to you. It forms the core of the computer science field. The methods that you have at your fingertips and deal with everyday in the normal course of software design are perfectly designed to do this. So, if only you have the interest, you do have the capacity and you do have the means" (Alexander, 1996).

References

Alexander, C., Ishikawa, S. & Silverstein, M. (1977). *A Pattern Language: Towns, Buildings, Construction*, Oxford University Press.

Alexander, C. (1979). *The Timeless Way of Building*, Oxford University Press.

Alexander, C. (1996). *The Origins of Pattern Theory, the Future of the Theory, And the Generation of a Living World*, OOPSLA'96, SIGPLAN.

Alexander, C. (2002). *The Nature of Order: an essay on the art of building and the nature of the universe*, Center for Environmental Structure.

Alexander, C. (2010). *An interview with Christopher Alexander*, Transition Culture.

Beck, K. & Cunningham, W. (1987). *Using Pattern Languages for Object-Oriented Programs*, OOPSLA'87, SIGPLAN.

Beck, K., Grenning, J., Martin, R.C., Beedle, M., Highsmith, J., Mellor, S., van Bennekum, A., Hunt, A., Schwaber, K., Cockburn, A., Jeffries, R., Sutherland, J., Cunningham, W., Kern, J., Thomas, D., Fowler, M. & Marick, B. (2001). *Manifesto for Agile Software Development*, Agile Alliance.

Curl, J.S. (2018). *Making Dystopia*, Oxford University Press.
Darwin, C. (1859). *On the Origin of Species*, Charles Murray.
Duany, A., Plater-Zyberk, E. & Speck, J. (2001). *Suburban Nation: The Rise of Sprawl and the Decline of the American Dream*, Farrar Straus & Giroux.
Eiben, A.E. & Smith, J.E. (2003). *Introduction to Evolutionary Computing*, Springer.
Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.
Leitner, H. (2015). *Pattern Theory: Introduction and Perspectives on the Tracks of Christopher Alexander*, CreateSpace.
Leuf, B. & Cunningham, W. (2001). *The Wiki Way*, Addison-Wesley.
Mehaffy, M. (2011). *Horizons of Pattern Languages*, CNU2011.
Mehaffy, M.W. & Salingaros, N.A. (2015). *Design for a Living Planet: Settlement, Science and the Human Future*, Sustasis Press.
Postle, B. (2013). *An Adaptive Approach to Domestic Design*, JBU III, Journal of Biourbanism.
Postle, B. (2018). *Homemaker: Free Software project*, Bitbucket.
Salingaros, N.A. (2017). *Design Patterns and Living Architecture*, Sustasis Press.