

Colorado Technical University



Technical Report Computer Science

Software Architecture and the Use of Patterns: How Christopher Alexander's *The Timeless Way of Building* can be applied to Software Design

Sydney Caddel

**Technical Report Number
CTU-CS-2001-06**

Software Architecture and the Use of Patterns: How Christopher Alexander's *The Timeless Way of Building* can be applied to Software Design

Sydney Caddel¹

Abstract.

Christopher Alexander's *The Timeless Way of Building* is summarized and its application to software design and software patterns is discussed.

The field of software architecture has been influenced by some of the ideas from building architecture and indeed there are some parallels. One area of growth is the concept of pattern languages in the design and development of software systems. The pattern language movement bases its philosophy on the work of architect Christopher Alexander. Beginning in 1979, Alexander wrote a series of three books describing the use of patterns and pattern languages for designing individual buildings as well as entire towns. Norman Kerth and Ward Cunningham discuss Alexander's "quality without a name" concept as an element in software architecture and in fact, several of the course articles reference Alexander's work². I do not have experience in industry, so cannot base a term paper on real world experiences. I chose, however, to read the first of Alexander's books, *The Timeless Way of Building*³, to gain a better understanding of the foundation of the pattern movement and to determine how closely it applies Alexander's theory.

Alexander begins *The Timeless Way of Building* with the statement, "The books are intended to provide a complete working alternative to our present ideas about architecture, buildings and planning – an alternative which will, we hope, gradually replace current ideas and practices⁴." Judging from the articles assigned for class, however, the software architecture pattern movement has not taken Alexander's theory as a whole, nor taken it far enough to produce software in the manner Alexander suggests for producing buildings. In defense of the software architecture community, it is important to remember that even building architects have not yet fully accepted applied Alexander's theory, as seen in the vast quantity of build-

¹ Edited by Bo Sandén.

² These articles are, Clements and Northrup, "Software architecture: An executive overview", Kerth and Cunningham, "Using Patterns to improve our architectural vision", Coplien, "idioms and patterns as architectural literature" and Cockburn, "The interaction of social issues and software architecture".

³ Alexander, *The Timeless Way of Building*, New York: oxford University Press 1979.

⁴ Alexander, *Timeless*, front piece

ings lacking quality. *The Timeless Way of Building* offers a philosophy applicable to software architecture, a philosophy that is, as Alexander suggests, an alternative to our present ideas about architecture, building (implementation) and planning of software systems. Software architecture, for the purposes of this paper is the design and development of software systems of any size. This is parallel to building architecture, in which architecture encompasses all aspects of design and development – requirements gathering, design concept, design drawings and models, and the working drawings or blueprints used to build the actual building. Design concepts are the first thoughts about a design for the software – what will be the driving force behind this software system, how will the end user interface with the system – in other words, the philosophy driving the software design.

An understanding of Alexander’s philosophy as he states it in *The Timeless Way of Building* is necessary before we can determine if the current software architecture pattern movement follows it. I briefly summarize Alexander’s main points: “quality without a name” and determining its presence; patterns and pattern languages; using pattern languages to create buildings (or software); repair (or maintenance) with pattern languages; and how the use of pattern languages allows us to create “whole” structures. Finally, I suggest that software engineers apply Alexander’s philosophy of design to software systems for a radical new approach to software design and development.

Alexander describes a timeless way of building which he explains is thousands of years old and yet the same today as it was centuries ago. It has created the most beautiful buildings in the world; buildings that are alive and produce that feeling within us. There is a method tied to the timeless way that is precise but cannot be applied mechanically. Alexander points out that we have become so rule driven that we are “afraid of what will happen naturally, and convinced that we must work within a ‘system’ and with ‘methods’ since without them our surroundings will come tumbling down in chaos.”⁵ The timeless way of building Alexander describes embodies the “quality without a name.” This quality is one that most people can readily identify, but is difficult to define. He explains that “quality without a name” is never the same twice because it takes its character from the places where it occurs; it is free from inner contradictions; it is alive, whole, comfortable, free, exact, egoless, and eternal; it has the character of nature; and places with this quality invite it to come to life in us.⁶

Buildings exhibit “quality without a name”, as patterns of events that keep happening and are tied to the space where they occur. Alexander’s example is the pattern of events that he calls “watching the world go

⁵ Alexander, *Timeless*, p. 14

⁶ Alexander, *Timeless*, pp 16, 27, 39, 143 and 53

by” and this pattern “cannot [be] separate[d] from the porch where it occurs”⁷. Alexander states that “on a geometric level, we see certain physical elements repeating endlessly, combined in an almost endless variety of combinations,”⁸ and “beyond its elements each building is defined by certain patterns of relationships among the elements.”⁹ It is this combination of recurring patterns of events that gives a building its character, a character that may be alive and produce good feelings, or be dead.

Alexander differentiates between “alive” and “dead” patterns. Alive patterns are capable of “resolving their own internal forces”, a Zen idea suggesting an innate character within an object such as a building. Saying that a pattern is alive, Alexander explains, is equivalent to saying that it is stable. He contrasts alive patterns with dead patterns, stating, “a pattern which prevents us from resolving our conflicting forces, leaves us almost perpetually in a state of tension.”¹⁰ Essentially, he is describing what many people have experienced – the feeling of frustration due to a building (or software system) made up of dead patterns. The quality produced by alive patterns cannot be *made* any more than a flower can be built, petal by petal, but must be *grown* like the flower from a seed. Combining patterns is the process that allows the architect to grow a building (or software system).

Patterns, according to Alexander, are rules of thumb that describe the objects they are meant to generate through their definition. A pattern language, then, is a system of patterns that allow its “users to create an infinite variety of ... combinations of patterns we call buildings, gardens, towns”¹¹ or even software systems. In its simplest form, a pattern language is made up of two sets, a set of elements and a set of rules for combining them. This is very similar to the grammars computer scientists use to define computer languages. Alexander is convinced that these pattern languages exist in every person’s mind from the greatest artist to the humblest builder. He points out that while the rules are simple, this does not imply that they are simple to see or create. As an example, Alexander reminds the reader that even the “ugliest and most deadening places in the world are made from patterns.”¹² The problem facing the modern world is that the centuries old patterns and pattern languages have died, either through disuse or through a breakdown in communicating them. Chaos follows the breakdown of the pattern languages and “in panic, people try to replace the lost order of the organic process, by artificial forms of order based on control.”¹³ This artificial control only makes things worse, and can be seen in the many “planned” communities with

⁷ Alexander, *Timeless*, p. 70

⁸ Alexander, *Timeless*, p. 82

⁹ Alexander, *Timeless*, p. 85

¹⁰ Alexander, *Timeless*, p. 114

¹¹ Alexander, *Timeless*, p. 186

¹² Alexander, *Timeless*, pp. 202-03, 222, 228.

¹³ Alexander, *Timeless*, p. 237.

restrictive covenants, or in the forced order seen in the Capability Maturity Model (CMM) and the variety of document-centered models focusing on milestones and deliverables instead of quality and process. Alexander's conclusion is that this simple revelation regarding the lost organic order and its replacement with an artificial order "calls for a shattering revision of our attitude toward architecture and planning."¹⁴ The solution is to rediscover patterns and pattern languages so that we can begin again to create buildings, and I would argue software, that are alive and possess the quality without a name.

Before we can create buildings or software systems with patterns, Alexander argues that we must first find a way to communicate patterns such that people can understand them. Every pattern has three parts that express a relationship between the context (domain), the problem (purpose of the pattern for the software system) and a solution (application of the pattern in the implementation).¹⁵ The process of communicating a pattern begins by making its inner structure clear, which I would argue is easier said than done. Observation is the key to discovering patterns that are alive; those patterns that feel good have a common property, which is missing from those that feel bad. A good language, one that is capable of creating something whole, is "morphologically and functionally complete. ... It is morphologically complete when I can visualize the kind of buildings that it generates very concretely. ... It is functionally complete when the system of patterns it defines is fully capable of allowing all its inner forces to resolve themselves."¹⁶

An important step in this process, then, is to ensure that the pattern language is good. Alexander proposes some simple guidelines to aid in this process. First, we must make each pattern easily understood so that it can fit with the other patterns of the language. Second, you must be able to draw a diagram of the pattern. If you are unable to draw it, then it is not a pattern according to Alexander. And finally, each pattern must be named. Alexander argues that the real work of design lies in this process of creating the language because it is the "structure and content of the language" that determine the design. Once the language is complete then the particular design can be generated. A general language has the ability to make a single building live also has the ability to make a thousand buildings live.¹⁷ In essence, Alexander is describing reuse as it is understood in object oriented design; creating a language (or a class) general enough to be used over and over again to create many buildings instead of just one. Not only must the language be general enough for use, but to be living, it must exist in every person in society. Once the language is shared it will evolve as people exchange ideas and patterns, and the inventory of patterns in the pattern pool will

¹⁴ Alexander, *Timeless*, p. 240.

¹⁵ Alexander, *Timeless*, p. 247.

¹⁶ Alexander, *Timeless*, p. 317.

¹⁷ Alexander, *Timeless*, p. 267, 324.

continue to change. Alexander argues that “it is essential that people ... shape their surroundings for themselves.”¹⁸ This is because design is not according to Alexander, a process of synthesis since it is impossible to create an object that has the quality without a name from preformed parts; it is only possible to “make a place alive by a process in which each part is modified by its position in the whole.”¹⁹ The process of the parts being modified by their position in the whole system leads to Alexander's discussion of the design process itself.

Alexander describes a design process in which an individual can create a building in the mind simply by letting a sequence of patterns generate it on the site. He describes how the same process can work with a group of people communicating with a common pattern language to design a larger building. This philosophy of design is especially apropos for software architecture given the large scale of modern software systems and that these large systems are designed and built by teams rather than a single individual. Alexander also argues that design should be carried out on the site saying, “the details of a building cannot be made alive when they are drawn at a drawing board ... because these drawings always assume, for the sake of simplicity, that the various manifestations of a given part are all identical.”²⁰ In other words, without a complete understanding of the site (or domain), many important details are assumed to fit a generic mold, creating a poor fit when the design is finally placed in its actual location.

A final stage in Alexander's timeless way of building is a process he calls repair, which software engineers would recognize as maintenance. Alexander suggests that instead of deteriorating the product, repair “generates an even more coherent whole, by making sure that every act contributes to the order of the previous acts.”²¹ Repair is necessary because “no building is ever perfect,”²² nor is any software system. Repair should be carried out based on the real events that actually happen. What Alexander calls repair sounds very much like software maintenance; no software is ever perfect so it has to be modified to accommodate real circumstances in its domain. Perhaps his philosophy could be applied to this area of software development.

Alexander's philosophy of design is radical for building architecture and even more radical for software systems architecture. Much of what he describes seems disconnected and leads to questions such as, can structure emerge from the spontaneous interaction of parts, or do we need a plan or blueprint? Alexander's

¹⁸ Alexander, *Timeless*, p. 354.

¹⁹ Alexander, *Timeless*, p. 369.

²⁰ Alexander, *Timeless*, p. 461.

²¹ Alexander, *Timeless*, p. 479.

²² Alexander, *Timeless*, p. 479.

response to these questions is to remind the reader of the early days of biology when the question was “how is an organism formed?”²³ Alexander believes that his philosophy allows a whole to emerge that takes on an “ageless character which gives the timeless way its name. ... This character is ... the physical embodiment ... of the quality without a name.”²⁴

As I read *The Timeless Way of Building*, I found that I could easily replace the words “building” and “town” with “software” and “system”, respectively. This is because there are parallels in terminology as well as philosophy. Application of this philosophy would place the location of our initial design work (requirements gathering as well as design) in the end user's environment. It would also require full participation of the end user in the entire design process. This would mean more than memos and meetings to have the user sign off; it would mean actually involving the user in making design decisions. Alexander’s concept of recurring patterns of events also suggests a greater emphasis on event-driven software systems. If we applied Alexander's entire philosophy to software architecture, we would witness a radical shift in the process of designing software systems.

²³ Alexander, *Timeless*, p. 497.

²⁴ Alexander, *Timeless*, p. xv