Chapter 20. Lesson 5, Reading 3
Darwinian Processes and Alexandrine Patterns
By Nikos Salingaros


Darwinian design

Elsewhere (Salingaros, 2006), I proposed that all good design is necessarily adaptive, and that the optimal method of achieving an adaptive design follows a Darwinian process. By this I mean an evolution of a group of similar competing design solutions for a particular project, of which the most adaptive is selected in stages. This process requires a set of selection criteria that are used as the basis of selection or “culling” from amongst the various alternative design choices that are generated. I am describing an intentional procedure, not to be confused with an entirely random proliferation devoid of selection.

As in biological evolution, the selection criteria strongly influence what the final result looks like. Therefore, a set of selection criteria based on adaptivity will generate an adaptive design; whereas a set of criteria based on comparison to certain prototypes will guarantee that the end result will resemble that reference prototype. In biology, adaptive selection to different environments has taken place to produce what we today regard as entirely different animals — starting from the same common ancestor.

Selection via comparison to a prototype is not necessarily bad, if the prototype itself is adapted to the uses of the required design solution. This can happen only if the prototype has been produced by evolutionary adaptation. While the end result of copying a prototype may not be the most original possible, it does guarantee a strong measure of usefulness, as the derived design inherits the adaptive properties of the original. This is the method of traditional design: copy a set of prototypes, which themselves have evolved by selection over millennia to adapt to particular uses, and the end result is guaranteed to be adequate. The only problem arises if local forces are not accommodated by the prototype.
Copying a prototype leads to disaster when that prototype has not evolved, but is imposed (i.e. is defined *ad hoc*). This occurred on a massive scale during the twentieth century, when arbitrary geometrical forms were presented as architectural and urban prototypes. Those prototypes were based on abstract reasoning that itself had only a tenuous connection to social and philosophical concepts; none of which relates to human activities, functions, or sensibilities. Matching to those prototypes produces non-adaptive designs that never achieve any degree of user comfort, either physically or psychologically (Salingaros, 2006).

Nevertheless, twentieth-century design based on matching to simple abstract prototypes was extraordinarily successful, because it was very easy to use in practice (Salingaros, 2006). This is demonstrated by counting the number of steps in the design method. We can estimate very roughly the number of steps in an intentional Darwinian processes corresponding to selections in each of the above-mentioned methods. In order-of-magnitude estimates: (1) modernist design requires only a few (usually less than five) steps to match pure geometric prototypes; (2) traditional vernacular design, including Classical, typically requires on the order of twenty to thirty steps to match traditionally-derived prototypes; (3) an innovative adaptive design that is not anchored to any traditional form may in general require up to one-hundred steps to evolve its adaptations.

Strictly in terms of economy in the number of design steps — which corresponds to hard mental effort at developing design variants, and choosing the most appropriate ones among them — modernist design wins out over any other design method. So, we have to recognize its tremendous advantage of economy. This economy in turn helps to explain modernism’s widespread adoption during the twentieth century. To replace modernist design with an adaptive design method, one has to be convinced of the benefits of such a change.

**Sorting algorithms as an analogy for design**

An excursion into computing will serve to illustrate two basic approaches to design: (i) intentional, top-down design, versus (ii) evolved, bottom-up design. It will also help us to understand adaptivity. I claim that both techniques can be made to work to achieve a final product that is of comparable utility. I will then argue that they are equivalent in an abstract mathematical way. The central question of design adaptivity will only be addressed afterwards.

A recent result in computer science has important implications for design. This result is not widely known in architectural or urbanist circles, so I present it here. It is useful to consider design as an algorithm: a set of instructions to be followed in order to achieve a particular result. There are deep connections between architecture and computer science, which first became obvious in the success that the “patterns” introduced by Christopher Alexander in architecture (Alexander *et al.*, 1977) eventually had in software.

The present discussion is distinct from design patterns. One of the simplest possible programs is a number-sorting program, which takes a list of numbers and
sorts them into increasing magnitude. The reason such a program is so simple is that its generative components — corresponding so to speak to the DNA in a biological entity — are basically two: comparing, and switching. There are instructions to compare two numbers to see which is greater, and other instructions to either leave these two numbers in their original order, or to switch them. By a judicious combination of comparing and switching instructions, one creates a number-sorting algorithm.

In the specific example to be discussed here, a list of 16 numbers is sorted. This is known as a “sorting network for \( n = 16 \)”. It became something of a challenge for the smartest computer programmers to write the shortest (i.e. optimal) program that could sort a list of numbers. The shortest programs for this task were written using fewer and fewer exchanges as follows: with only 65 exchanges in 1962; 63 exchanges in 1964; 62 exchanges in 1969; and 60 exchanges in 1969 (Hillis, 1992). The question remains open whether it is possible to write an even shorter program to achieve the same task.

**Darwinian evolution of algorithms**

The computer scientist Danny Hillis developed a Darwinian setting for evolving number-sorting algorithms (Hillis, 1992; 1998). By generating an enormous variety of programs containing randomly-distributed switching components, he selected those that achieved some partial success in sorting number lists. (Actually, Hillis took the first 32 exchanges from the most successful existing programs for sorting 16 numbers, and allowed the number and character of all subsequent exchanges to evolve). He then combined those programs or introduced random shufflings in each one, and after each shuffling checked them for sorting ability. By doing this an enormous number of times on one of the most powerful computers (which he himself designed and built), Hillis was able to evolve a sorting algorithm starting from a random collection of basic components. The result was a sorting algorithm with only 61 exchanges.

The results are profound in their implication. First the obvious result: a Darwinian process evolved a program out of a mishmash of switching instructions, which is just as efficient as those developed by the best human minds. The second result was totally unexpected: *Hillis cannot understand how the evolved algorithm actually works* (Hillis, 1998). The 30 or so evolved exchanges are in a configuration that does not reveal a recognizable sorting pattern. It is reasonable to suppose, then, that it is unlikely the evolved sorting algorithm could have been written by a human programmer.

This single example demonstrates that a Darwinian process need not necessarily result in an understandable pattern. (This does not mean, however, that all the results of Darwinian selection cannot be understood). We can test such evolved algorithms to make sure they are correct and efficient, yet their internal complexity somehow escapes us. This was revealed in one of the simplest possible algorithms
— a sorting program for 16 numbers. Clearly, more complex systems are bound to have an even higher, and perhaps incomprehensible, complexity.

Human ingenuity using proven programming methods led to a program with 60 switches, whereas a Darwinian process led to a program with 61 switches. The results are almost exactly comparable in their efficiency. This suggests something about architectural design. Intentional, top-down design that is based on evolved prototypes can indeed be compared with evolved, bottom-up design. Both intentional and bottom-up approaches give optimal solutions of comparable fitness, while the results of an evolutionary approach are in a fundamental sense unexpected.