that are fabricated (often in a lengthy sequence of steps, each one of which adds value), acquired, transported to the required location, and finally fixed into place within an assembly. From a building maintenance perspective, they are replaceable parts. And from a functional perspective, they are modules that perform identifiable tasks within a building, and that can be composed with other modules to create subsystems that perform higher-level tasks. When modules are composed in this way, they not only have spatial relationships that may delight the eye, they also transfer something—structural loads, for example—across their interfaces by virtue of these relationships.

Similarly, programming languages provide ways of breaking code down into modules, and hierarchically assembling modules to produce higher-level modules and eventually complete software systems. As all programmers know, good code isn't a mess; it has a clear, logical structure of modules and hierarchies, implemented using the language's abstraction and organization constructs. The organizational clarity of classical architecture provides an excellent model for this.

With occasional exceptions, works of architecture follow additional principles of internal order as well. Columns lined up in a row, for example, are normally regularly spaced. If you want to write code to generate a CAD model of a colonnade, then you don't specify the location of each column individually. You employ iteration, with a location parameter that is incremented at each step. In other words, you express the principles of the architecture in a more concise and elegant way, and one that provides the reader of the code with more insight.

What if you want to generate a regular column grid? You employ nested iteration. First you iterate a column to generate a regularly spaced row of columns, and then you iterate the row of columns as many times as you want to create the grid.

What if you want to make the corner columns different from the internal columns? (Architects often do this in response to the different structural and other conditions that exist at the corners of buildings.) You use a conditional: *if* it's a corner condition, *then* substitute the alternative column design. If you want to vary the spacing of columns at the center to mark the importance of the central axis, distinguish external columns from internal columns, and so on, you just introduce additional conditionals.

Modularity, hierarchy, and regular repetition are by no means the only ordering principles commonly employed by architects. If you analyze architectural compositions carefully, you can often find regularities in dimensions and proportions, symmetries (and artfully broken symmetries), nested self-similar shapes as in fractals, and parametrically varied motifs. I leave it as an exercise for the reader to imagine the expression of these principles in code.

Sometimes, though, internal order seems to be lacking. An architect might, for some reason, scatter columns randomly. What then? It turns out that the most concise way to describe this sort of configuration is to specify the location of each column individually. There is no shorter, more economical description.