CS 335 — Software Development

Introduction to Design Patterns

Sept 5 and 7, 2007

Instead of being widely shared, the pattern languages which determine how a town gets made becomes specialized and private. Roads are built by highway engineers; buildings by architects; parks by planners. . .

The people of the town themselves know hardly any of the languages which these specialists use. And if they want to find out what these languages contains, they can't, because it is considered professional expertise. The professionals guard their language jealously to make themselves indispensable.

Alexander, *The Timeless Way of Building*, pg 231–232.

There were hundreds of people, each making his part within the whole, working, often for generations. At any given moment there was usually one master builder, who directed the overall layout. . . but each person in whe whole had, in his mind, the same overall language. Each person executed each detain in the same general way, but with minor differences.

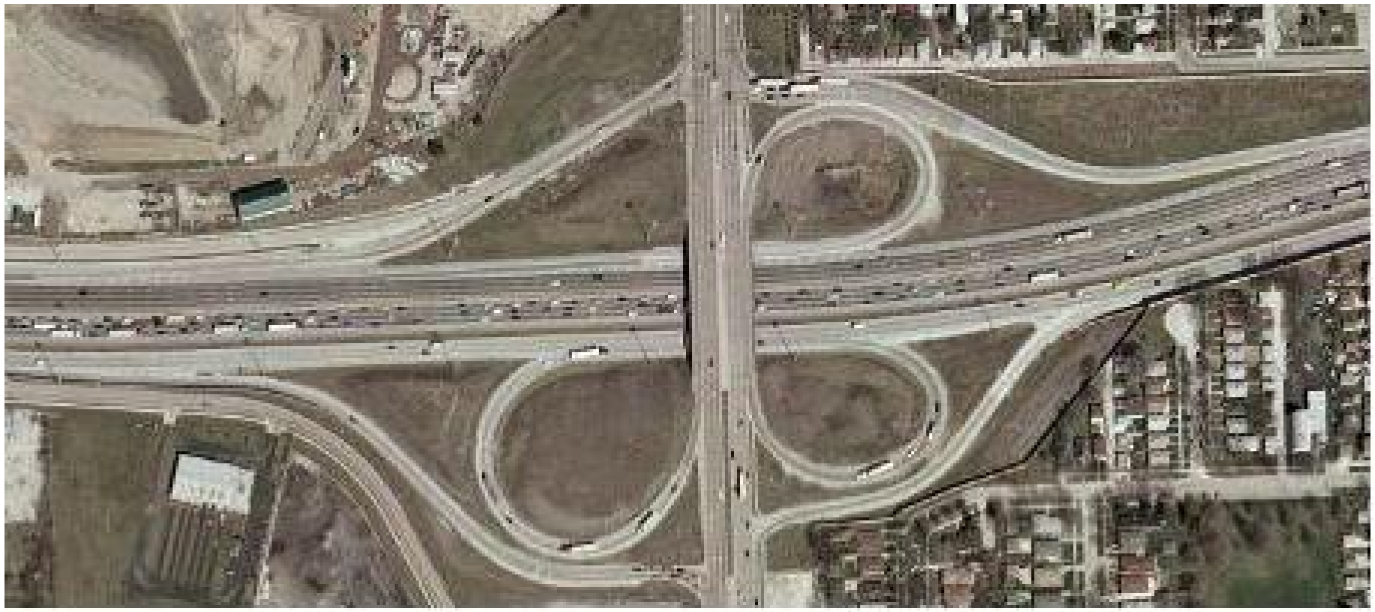
. . . The fact is that Chartres, no less than the simple farmhouse, was built by a group of men, acting within a common pattern language, deeply steeped in it of course. It was not made by "design" at the drawing board.

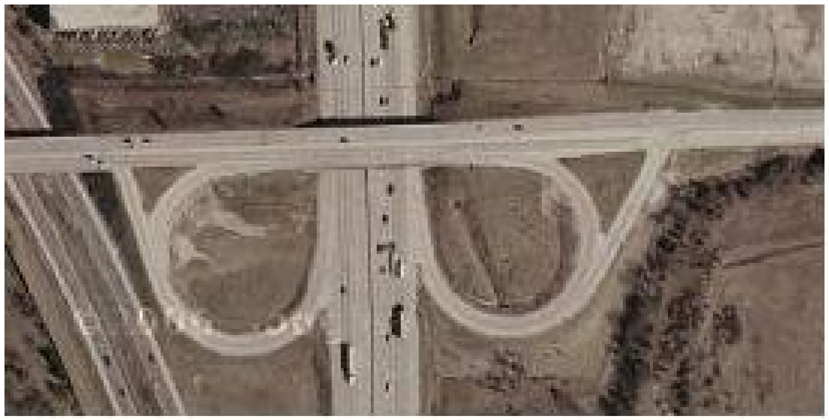
Alexander, *The Timeless Way of Building*, pg 216–217.

The elements of [an architectural pattern language] are entities called patterns. Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice.

Alexander, *A Pattern Language*, pg x.

Images from Google Maps.





Images from Google Maps.

S 380 W
231
231
65
S 380 W

Images from Google Maps.

Pattern 3: City Country Fingers

Problem: Continuous sprawling urbanization destroys life, and makes cities unbearable. But the sheer size of cities is also valuable and potent.

Solution: Keep interlocking fingers of farmland and urban land, even at the center of the metropolis. The urban fingers should never be more than 1 mile wide, while the farmland fingers should never be less than 1 mile wide.

Alexander, *A Pattern Language*, pg 21 - 25.

Pattern 37: House Cluster

Problem: People will not feel comfortable in their houses unless a group of houses forms a cluster, with the public land between them jointly owned by all the householders.

Solution: Arrange houses to form very rough, but identifiable clusters of 8 to 12 households around some common land and paths. Arrange the clusters so that anyone can walk through them, without feeling like a trespasser.

Alexander, *A Pattern Language*, pg 197 - 203.

Pattern 50: T Junctions

Problem: Traffic accidents are far more frequent where two roads cross than at T junctions.

Solution: Lay out the road system so that any two roads which meet at grade, meet in three-way T junctions as near 90 degrees as possible. Avoid four-way intersections and crossing movements.

Alexander, *A Pattern Language*, pg 264-265 - 25.

Pattern 111: Half-Hidden Garden

Problem: If a garden is too close to the street, people won't use it because it isn't private enough. But if it is too far from the street, then it won't be used either, because it is too isolated.

Solution: Do not place the garden fully in the front of the house, nor fully to the back. Instead, place it in some kind of half-way position, side-by-side with the house, in a position which is half-hidden from the street and half-exposed.

Alexander, *A Pattern Language*, pg 545 - 547.

Pattern 112: Entrance Transition

Problem: Buildings, and especially houses, with a graceful transition between the street and the inside, are more tranquil than those which open directly off the street.

Solution: Make a transition space between the street and the front door. Bring the path which connects street and entrance through this transition space, and mark it with a change of light, a change of sound, a change of direction, a change of surface, a change of level, . . . and above all with a change of view.

Alexander, *A Pattern Language*, pg 548–552.

Pattern 127: Intimacy Gradient

Problem: Unless the spaces in a building are arranged in a sequence which corresponds to their degrees of privateness, the visits made by strangers, friends, guests, clients, family, will always be a little awkward.

Solution: Lay out the spaces of a building so that they create a sequence which begins with the entrance and the most public parts of the building, then leads into the slightly more private areas, and finally to the most private domains.

Alexander, *A Pattern Language*, pg 610–613.

Pattern 136: Couple's Realm

Problem: The presence of children in a family often destroys the closeness and the special privacy which a man and wife need together.

Solution: Make a special part of the house distance from the common areas and all the children's rooms, where the man and woman of the house can be together in private. Give this place a quick path to the children's rooms, but, at all costs, make it a distinctly separate realm.

Alexander, *A Pattern Language*, pg 648 - 650.

Pattern 179: Alcoves

Problem: No homogenous room, of homogenous height, can serve a group of people well. To give a group a chance to be together, as a group, a room must also give them the chance to be alone, in one's and two's in the same space.

Solution: Make small places at the edge of any common room, usually no more than 6 feet wide and 3 to 6 feet deep and possibly much smaller. These alcoves should be large enough for two people to sit, chat, or play and sometimes large enough to contain a desk or a table.

Alexander, *A Pattern Language*, pg 828 - 832.

Problem

You have a large amount of data of the same type on which you need to perform the same operation.

Problem

You need to perform some operation on every item in an array.

Problem

You need the same functionality many times, in different contexts, at different parts of the program.

Problem

You have several classes which are subtypes of the same type. For one of their common operations, all classes implement the method using the same basic algorithm. However, some of them differ in the details of certain steps of the algorithm. You want to make it easy for the classes to share code for the steps that are common to all, but also easy for classes to change various steps.

Solution: Template Pattern

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method lets subclasses redefine certain steps of an algorithm without changing the algorithm's structure

Gamma et al., *Design Patterns*, pg 325.

Problem

You have a super class that implements an operation involving the instantiation of new instances. However, various subclasses will instantiate different classes at this point.

Solution: Factory Method

Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.

Gamma et al., *Design Patterns*, pg 107.

Problem

You have n classes that are different in structure and operations except that there is one certain operation that they all must implement. That operation could be implemented in m possible ways, and any of the n planned classes could use any of the m implementations. You do not want to write $n \times m$ classes to cover all the possible combinations.

Solution: Strategy

Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.

Gamma et al., *Design Patterns*, pg 315.

Problem

It's important for some classes to have exactly one instance. How do we ensure that a class has only one instance and that the instance is easily accessible?

Making a class's methods and fields static will prevent the "object" from being substituted polymorphically with other objects with the same interface.

A global variable makes an object accessible, but it doesn't keep you from instantiated multiple objects.

Adapted from Gamma et al., *Design Patterns*, pg 127.

Solution: Singleton

Ensure a class only has one instance, and provide a global point of access to it.

Make the constructor private, instantiate the single instance statically, and provide a static method which will return that instance.

Adapted from Gamma et al., *Design Patterns*, pg 127.

Criticism

Some researchers have suggested that a pattern is “a solution to a problem in a context,” citing Chris Alexander’s work in architecture. . . Here are some thoughts on this.

1. A pattern is a template, not a specific solution.
2. Alexander’s “pattern” theory remains unaccepted by his peers.
3. No dictionary supports his definition of the word “pattern.” . . .
4. Although “a solution to a problem in a context” is a compelling writing style—after all, nearly every sales letter follows it—that does not make an instance of that writing style a “pattern.”

Peter Coad, *Object Models*, pg xiv.

Criticism

[I]n the OO world you hear a good deal about ”patterns”. I wonder if these patterns are not sometimes evidence of [the need to make code transformations the compiler should do]. When I see patterns in my programs, I consider it a sign of trouble. . . Any other regularity in the code is a sign, to me at least, that I'm using abstractions that aren't powerful enough. often that I'm generating by hand the expansions of some macro that I need to write. . . .

Peter Norvig found that 16 of the 23 patterns in *Design Patterns* were “invisible or simpler” in Lisp.

Paul Graham, “Revenge of the Nerds”