W4156

Design Patterns

Agenda

Do we need to design unique solutions to every problem?

- ❏ Do we need to produce novel designs each time?
- ❏ Lessons from Building Architecture
- ❏ Core Aspects of a Pattern
- ❏ OO Patterns
- ❏ Design Pattern Examples

Novel Designs Each Time?

We are becoming more capable & rounded software engineers (requirements, testing, design, methodologies)

- We ***can*** design novel solutions to problems (though it ***takes time***)
- Wouldn't it be great if we could ***reuse*** 'good designs'
- ***But*** we are coding solutions for different problem domains
  (hotels vs finance vs computer games vs etc)
- Is some underlying commonality in the problems we face...?

Lessons from Building Architecture*

We build a *lot* of rooms.

Recurring problem: Where do we place the door(s)?

Volunteer?

*We maintain buildings architecture are a poor metaphor for software. We are just borrowing a sub-discipline

Lessons from Building Architecture*

Small Rooms

Big Rooms

1 door, 2 doors

Entrance Way

Lessons from Building Architecture*

Can we distill out some ‘rules/heuristics’?

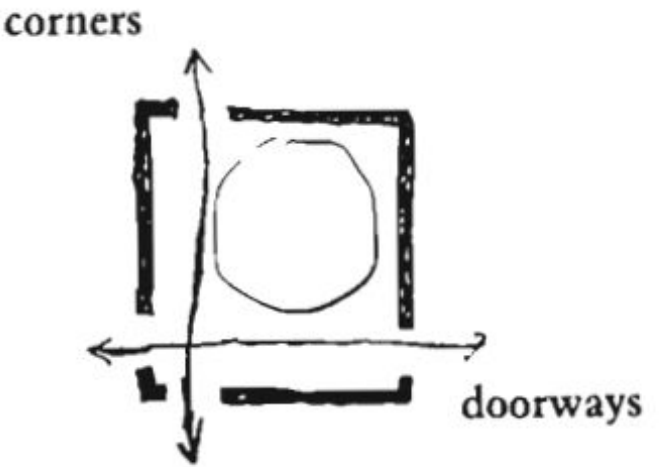
Can we apply these rules apply to different ***types*** of rooms?

196 CORNER DOORS*

The success of a room depends to a great extent on the position of the doors. If the doors create a pattern of movement which destroys the places in the room, the room will never allow people to be comfortable.

First there is the case of a room with a single door. In general, it is best if this door is in a corner. When it is in the middle of a wall, it almost always creates a pattern of movement which breaks the room in two, destroys the center, and leaves no single area which is large enough to use. The one common exception to this rule is the case of a room which is rather long and narrow. In this case it makes good sense to enter from the middle of one of the long sides, since this creates two areas, both roughly square, and therefore large enough to be useful. This kind of central door is especially useful when the room has two partly separate functions, which fall naturally into its two halves.

Rooms with one door.

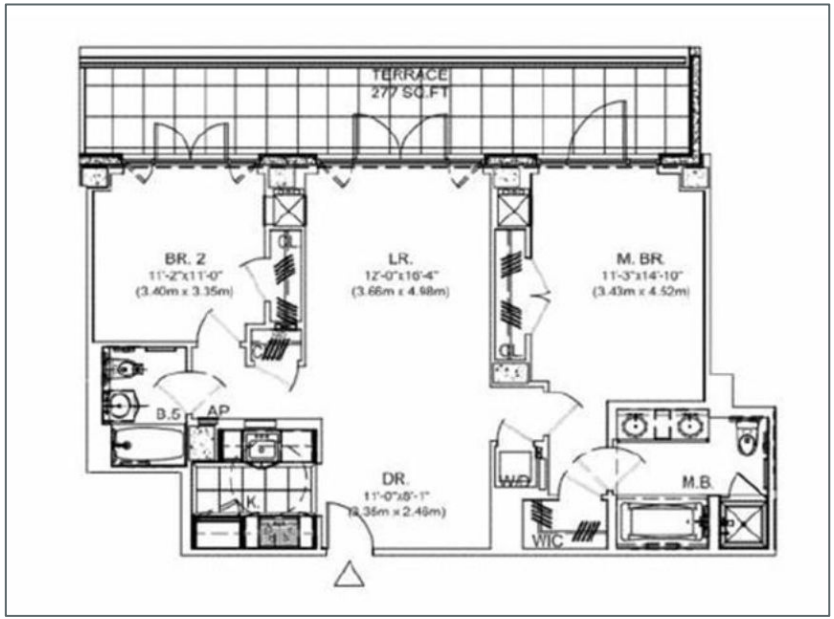


Door Placement

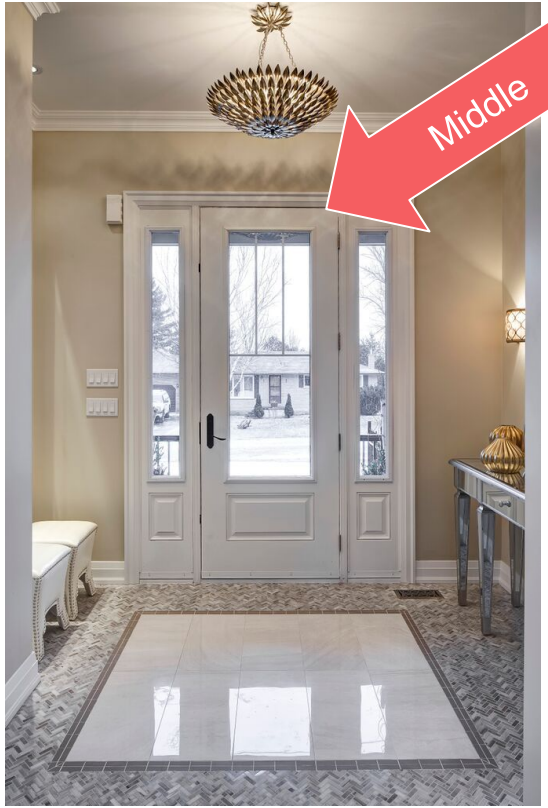
Except in very large rooms, a door only rarely makes sense in the middle of a wall. It does in an entrance room, for instance, because this room gets its character essentially from the door. But in most rooms, especially small ones, put the doors as near the corners of the room as possible. If the room has two doors, and people move through it, keep both doors at one end of the room.

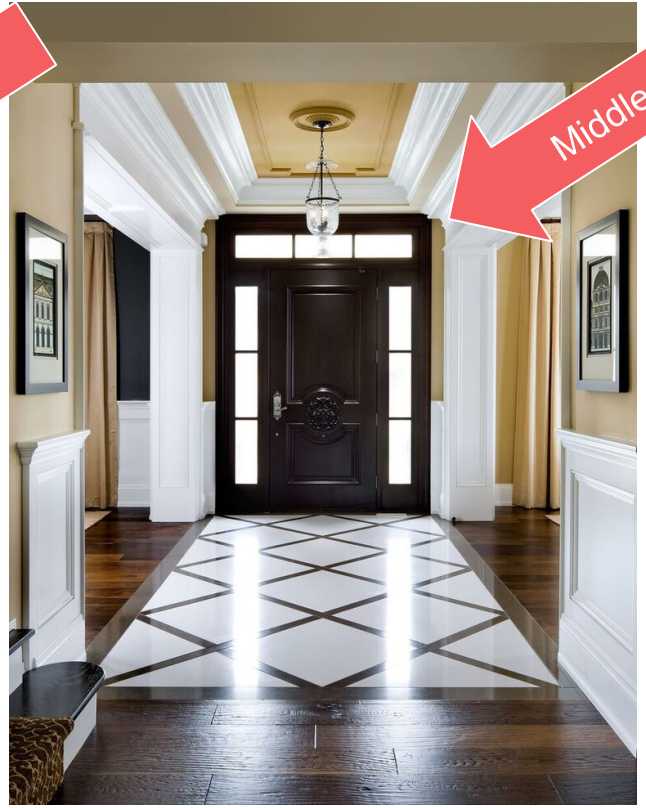
A Pattern Language - Alexander

Lessons from Building Architecture*

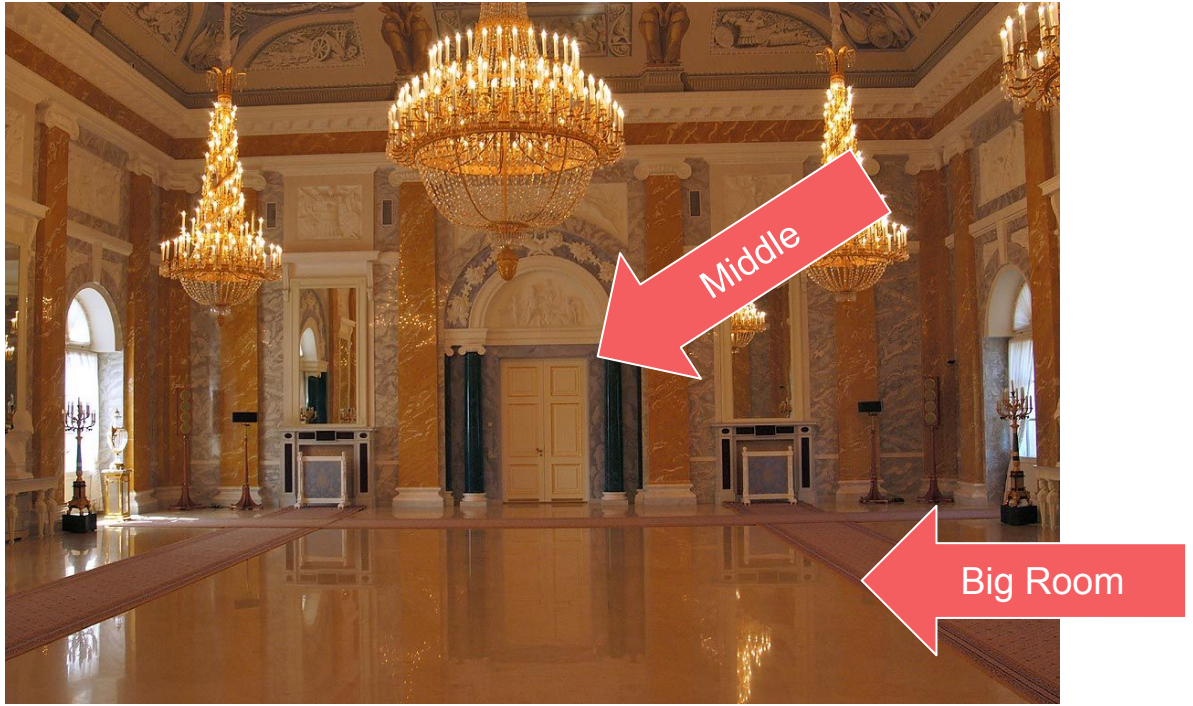


Lessons from Building Architecture





Lessons From Building Architecture



Lessons from Building Architecture

Patterns Defined

*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.* -- ChristopherAlexander

Applied at different levels: town, buildings, spaces.

Pattern Defined

Pattern Name	Door Placement
Problem (what problem do we encounter where the pattern is useful)	In rooms of different sizes, shapes, purposes and number of required doors - where do we place the doors?
Solution (describe pattern in terms of elements, relationships, responsibilities and collaborations)	Small rooms - in corners. If multiple doors then place in same corner Larger rooms - middle *may* make sense except Entrance Ways: Middle
Consequences (results and tradeoffs)	For smaller rooms we maximise the space and avoid bisecting rooms with space to transit between doors For grander entrance ways we create impact (though slower to cross through the room)

A pattern is ***not*** a concrete solution (a blue door for a living room in Poughkepsie)

Alexander Keynote Speech @ OOPSLA ‘96

My association with you—if I can call it that—began, oh it must have been two or three years ago. I began getting calls from computer people. Then somebody, a computer scientist, called me and said that there were a group of people here in Silicon Valley that would pay $3000 to have dinner with me.

I thought—what is this? It took me some time to find out. I didn't really understand what had been going on, and that my work had somehow been useful to computer science. Only now I'm now beginning to understand a little bit more of what you are doing in your field and the way in which it comes, in part, from some of the things that I've done. - Christopher Alexander

http://www.patternlanguage.com/leveltwo/archivesframe.htm?/leveltwo/../archives/ieee/ieee.htm

Returning to Software Engineering

1. Although we code solutions to different problem domains
2. However, we incur same repeating problems
3. There is a ‘catalogue’ of good design patterns which are reusable outlines
4. Our process
    a. ***Identify*** the problem we are facing we can
    b. ***Match*** the problem with a pattern in the catalogue
    c. **Apply** the pattern to our situation
    d. **Benefit** from a higher quality, reusable design and common language with other developers

OO Pattern Examples

There are many. We will drill into a few.
The more complex design patterns require deeper understanding of OO principles and constructs

Creational Patterns	Factory (there are a few variants), Python Object Pool, Java Singleton (though this is arguably an anti-pattern)
Structural Patterns	Decorator, Java, Python Adapter MVC
Behavioral Patterns	Command Observer Strategy

Pattern Defined (Laymans)

Pattern Name	Factory
Problem (what problem do we encounter where the pattern is useful)	We need to make a ‘Foo’ but it requires a series of steps to create, configure, initialize. (Foo in turn may have dependencies which also need to be created). This ‘creational’ logic pollutes our business logic which just wants to use Foo.
Solution (describe pattern in terms of elements, relationships, responsibilities and collaborations)	Create a class which handles all of the creation, configuration and ‘wiring’ together of the object it needs to make. The caller asks the factory for the object then receives a premade Foo
Consequences (results and tradeoffs)	Abstracts caller from creational logic. If factory returns an interface we can hide caller from concrete implementation

Remember: Pattern not a concrete solution

Pattern Defined (Laymans)

Pattern Name	Object Pool
Problem (what problem do we encounter where the pattern is useful)	We have some Foos (often database connections or other resources) that are expensive in time to create. Our code needs to use these object for a very short period of time. Creating the Foo, using it and then destroying it is slow.
Solution (describe pattern in terms of elements, relationships, responsibilities and collaborations)	Create an 'ObjectPool'. At startup (or dynamically) populate the ObjectPool with N Foo objects When your code wants a Foo it *asks* the pool for a Foo, uses it then *returns* the Foo to the Pool.
Consequences (results and tradeoffs)	More responsive code or better resource footprint as resources are pooled Caller must remember to return object to pool (even if exception thrown) so pool does not 'develop a leak'

Pattern Defined (Laymans)

Pattern Name	Decorator
Problem (what problem do we encounter where the pattern is useful)	We have an object Foo with specific behavior We want to enrich the behavior of Foo but do not want to change behavior for subtypes
Solution (describe pattern in terms of elements, relationships, responsibilities and collaborations)	Create a FooDecorator which has a Foo Caller calls FooDecorator which adds the new behavior before/after calling Foo
Consequences (results and tradeoffs)	Able to enrich the behavior of Foo Given FooDecorator has the same interface as Foo we can add or remove the decorator Allows Foo to maintain single responsibility

Interaction with R_____

Again - code does not get written in one linear sequence. It evolves both within an iteration and across iterations.

- If as you write the code you see the pattern then write it first time as a pattern
- You may see the pattern later. In which case you can r_____ the code into the pattern

Pop Quiz

Question	Answer
Patterns are concrete solutions to any given problem [T/F]	
Patterns represent reusable designs that can be applied to different situations [T/F]	
Patterns are design language between developers [T/F]	
Patterns can improve readability of code [T/F/Maybe]	
Correct application of Patterns can improve abstraction, encapsulation, testability and maintainability of code [T/F/Maybe]	

Reading

Reading	Optionality
Book in Python Code in Java Code in Python	Understand at least 1 design pattern from creational, behavioral, structural
GoF Book	Optional