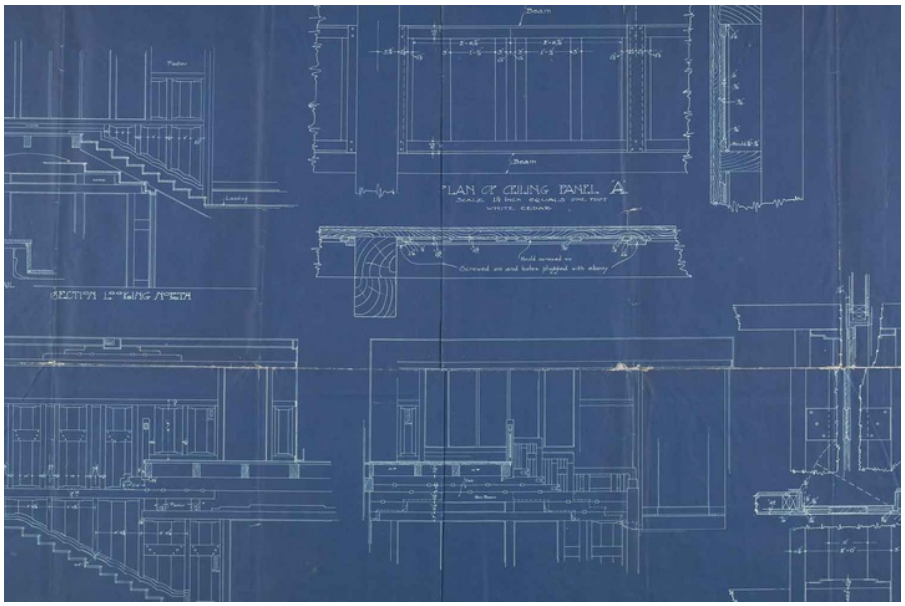
# Contrasting architecture patterns with design patterns

How architecture and design patterns can add clarity and understanding to your project.

By Neal Ford. September 30, 2015




Three elevations of a ceiling panel
(source: Ashley Van Haeften on Flickr)

Developers are accustomed to *design* patterns, as popularized in the book *Design Patterns* by Gamma, et al. Each pattern describes a common problem posed in object-oriented software development along with a solution, visualized via class diagrams. In the Software Architecture Fundamentals workshop, Mark Richards & I discuss a variety of *architecture* patterns, such as Layered, Micro-Kernel, SOA, etc. However, architecture patterns differ from design patterns in several important ways.

## Components rather than classes

Architectural elements tend towards collections of classes or modules, generally represented as boxes. Diagrams about architecture represent the loftiest level looking down, whereas class diagrams are at the most atomic level. The purpose of architecture patterns is to understand how the major parts of the system fit together, how messages and data flow through the system, and other structural concerns.



Architecture diagrams tend to be less rigidly defined than class diagrams. For example, many times the purpose of the diagram is to show one aspect of the system, and simple iconography works best. For example, one aspect of the Layered architecture pattern is whether the layers are *closed* (only accessible from the superior layer) or *open* (allowed to bypass the layer if no value added), as shown in Figure 1.

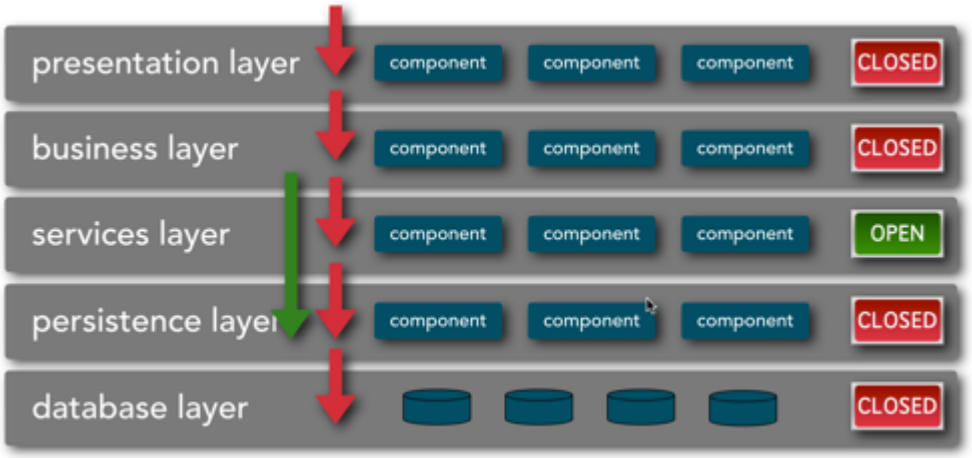



Figure 1: Layered architecture with mixed closed and open layers

This feature of the architecture isn't the most important part, but is important to call out because if affects the efficacy of this pattern. For example, if developers violate this principle (e.g., performing queries from the presentation layer directly to the data layer), it compromises the separation of concerns and layer isolation that are the prime benefits of this pattern. Often an architectural pattern consists of several diagrams, each showing an important dimension.

## Component types

Generally, design patterns utilize one type of entity (the class from object oriented programming) and illustrate the relationship between the classes within a solution.

Architecture patterns utilize a variety of component types, each typically composed of successively smaller modules. Each component has a responsibility within the architecture. For example, consider the diagram in Figure 2 of the Micro-kernel architecture pattern:

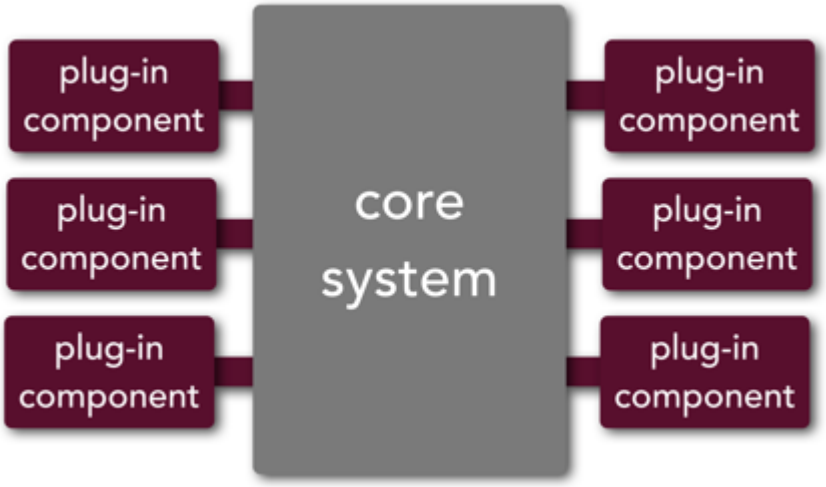



Figure 2: Component Types in micro-kernel architecture

Micro-kernel is a simple architecture with just two component types, *core* and *plug-in*. Other architecture patterns have many more moving parts; consider the space-based architecture pattern shown in Figure 3.

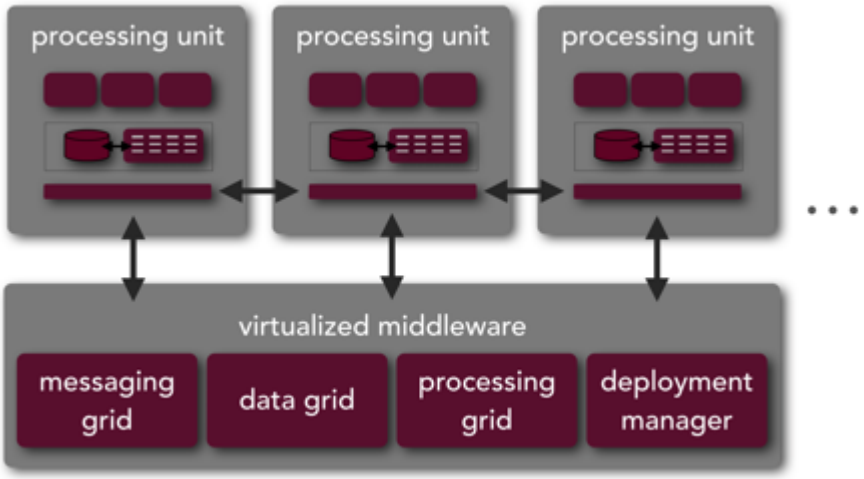



Figure 3: Component types in space-based architecture

The space-based architecture pattern includes 5 distinct components types:




| | |
|---|---|
| processing unit | where work is performed |
| messaging grid | manages input request and session |
| data grid | manages data replication between processing units |
| processing grid | manages distributed request processing |
| deployment manager | manages dynamic processing unit deployment |

*Table 1: Components types in a space-based architecture.*

Space-based architectures may have hundreds or thousands of processing units (indicated by the ellipsis in the diagram) and one each of the other component types. Each component has distinct responsibilities, making it important to delineate each separately.

## Variants and hybrids

Another common occurrence in architecture patterns are variants and hybrids. The *open* layer shown in Figure 1 is a good example of a traditional layered architecture variant.

Similarly, event driven architectures typically have two common variants, the *mediator* and *broker* patterns.

A good example of a hybrid architecture is the *service-based architecture*, a hybrid of the highly partitioned Service Oriented Architecture pattern and the bounded context of the micro-service pattern, as shown in Figure 4.

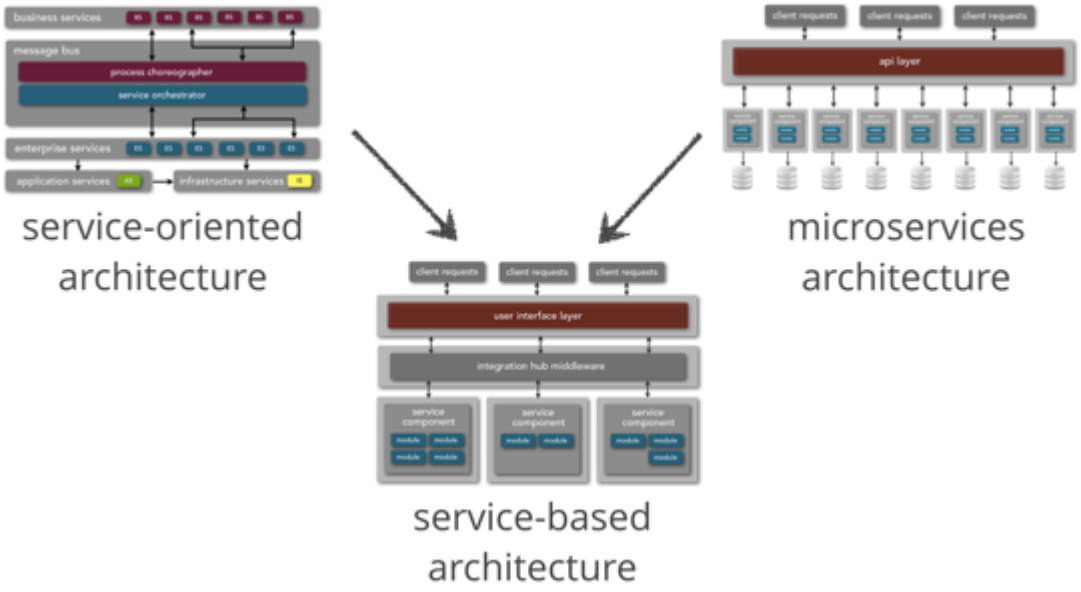



Figure 4: A service-based architecture is a hybrid between SOA and microservices.

It is also common to embed architectural styles within others. For example, in the service-based architecture in Figure 4, one of the services could be implemented using a micro-kernel architecture while another might use a layered approach.

Hybrids and variants are as common in architecture patterns as they are rare in design patterns. Because architecture represents practical building blocks, pragmatic variations abound.

## Scaffolding versus design

Ultimately, the differences between design and architecture patterns reflect their different uses. Architecture represents scaffolding, the frameworks that everything else sits upon. Design patterns represent a way to structure classes to solve common problems. While both are designed to add clarity and understanding, they operate at different levels of abstraction.

Architects must have both kinds of patterns at their disposal: design patterns to build the best internal structure, and architectural patterns to help define and maintain the underlying scaffolding of the application.

Article image: Three elevations of a ceiling panel (source: Ashley Van Haeften on Flickr).

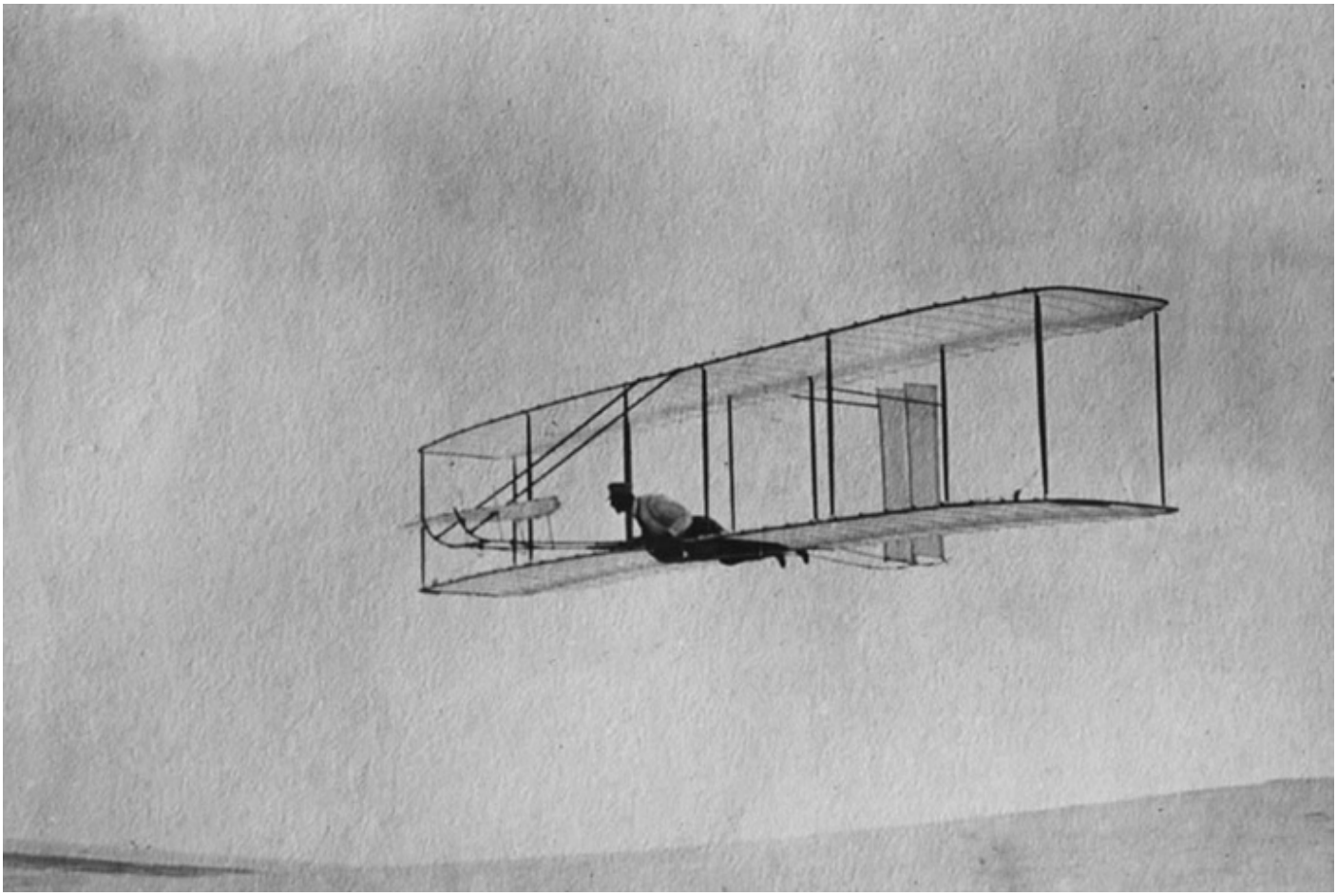


## Neal Ford

Neal is Director, Software Architect, and Meme Wrangler at ThoughtWorks, a global IT consultancy with an exclusive focus on end-to-end software development and delivery. Before joining ThoughtWorks, Neal was the Chief Technology Officer at The DSW Group, Ltd., a nationally recognized training and development firm. Neal has a degree in Computer Science from Georgia State University specializing in languages and compilers and a minor in mathematics specializing in statistical analysis. He is also the designer and developer of applications, ins...

more

SOFTWARE ARCHITECTURE



## Walking the tightrope: Balancing bias to action and planning

By Dianne Marsh

Dianne Marsh explains how microservices paved the way for traffic steering at Netflix, and she highlights current challenges.

SOFTWARE ARCHITECTURE

## Confessions of an enterprise architect

By Scott Shaw

Faced with business and regulatory complexity, Scott Shaw found himself committing some of the software development acts he once condemned. He confesses those sins and explains why they're sometimes necessary.

SOFTWARE ARCHITECTURE