Beautiful architectures find ways to localize information and behavior. At runtime, this manifests as *layering*, the notion that a system may be factored into layers, each representing a *layer of abstraction* or *domain*.

Automatic propagation

One fact in one place sounds good, but for efficiency's sake, some data or behavior is often duplicated. To maintain consistency and correctness, propagation of these facts must be carried out automatically at construction time.

Beautiful architectures are supported by construction tools that effect *meta-programming*, propagating one fact in one place into many places where they may be used efficiently.

Architecture includes construction

An architecture must include not only the runtime system, but also how it is constructed. A focus solely on the runtime code is a recipe for deterioration of the architecture over time.

Beautiful architectures are *reflective*. Not only are they beautiful at runtime, but they are also beautiful at construction time, using the same data, functions, and techniques to build the system as those that are used at runtime.

Minimize mechanisms

The best way to implement a given function varies case by case, but a beautiful architecture will not strive for "the best." There are, for example, many ways of storing data and searching it, but if the system can meet its performance requirements using one mechanism, there is less code to write, verify, maintain, and occupy memory.

Beautiful architectures employ a minimal set of mechanisms that satisfy the requirements of the whole. Finding "the best" in each case leads to proliferation of error-prone mechanisms, whereas adding mechanisms parsimoniously leads to smaller, faster, and more robust systems.

Construct engines

If you wish to build brittle systems, follow Ivar Jacobson's advice and base your architecture on use cases and one function at a time (i.e., use "controller" objects). Extensible systems, on the other hand, rely on the construction of virtual machines—engines that are "programmed" by data provided by higher layers, and that implement multiple application functions at a time.

This principle appears in many guises. "Layering" of virtual machines goes back to Edsger Dijkstra. "Data-driven systems" provide engines that rely on coding invariants in the system, letting the data define the specific functionality in a particular case. These engines are highly reusable—and beautiful.