Principles of System Architecture

OS12

Deniz Aydemir

Good architecture is a navigable map

Principle 1

Tag	All models are wrong but some are useful. (George Box)
Descriptive	A good system architecture serves as a legible and navigable map for its users.
Prescriptive	Compress unnecessary detail but provide enough info to allow users to navigate.
Description	A good architecture must function as a map for navigating a system. The actual system is the territory that exists in real life, and a good architecture must do the work of compressing the information in the system into a legible and communicable map. A map must first be legible. This means that the map must not over-compress the information of the system. It must also exist in a form that can be understood by those who will implement a system, whether that is in a diagram, a document, a software representation, or some other form. Like the map of a territory, a good architecture must be usable as a form of navigation for those who wish to build or maintain the system. A map must also be convertible to other maps. This means that the map should support conversions to new architectures, as required by those who may want to update the system.
Historical Reference	Albert Korzybski wrote about the ideas of maps and territories in mathematical semantics in his 1933 paper "Science and Sanity: An Introduction to Non-Aristotelian Systems and General Semantics". There have since been many versions of ideas like this. Similarly, this principle is a loose interpretation of the idea that a system could be represented as a Category in Category Theory. In Category Theory, a functor is the morphism of one category to another, and thus leads to the idea that a well-structured architecture should behave like a category and be "functorizable".
Sources	The Map is Not the Territory, Farnam Street Rationality, Eliezer Yudkowsky

Optimize mental resource allocation

NOVEL

Principle 2

Tag	Do that which consists in taking no action and order will prevail. (Tao Te Ching)
Descriptive	A good system architecture guides its users to allocate mental resources efficiently giving the user trust in the architecture as driver of the user's focus.
Prescriptive	Efficiently distribute the architecture user's effort and attention by creating friction where the system is most fragile and the risks are highest.
Description	A system architecture is a representation of the system that exists or will one day exist in real life. For that system architecture to be good, it must demand the mental resources of a user of that architecture. This means that it must sufficiently compress excessive details while avoiding over-compressing information that requires attention or is of greater importance to the system. In other words, a good system architecture will require the user of the architecture to devote more mental bandwidth and focus to the parts of the architecture that have higher stakes or are more fragile. When a system architecture achieves the proper balance of friction and seamlessness in the right places, then the user of the architecture can trust that architecture to sufficiently direct them, and the user's mental burden can be offloaded. The user trusts that they do not need to create extra friction for themselves when things need to be double checked, the architecture will provide the appropriate amount of friction for them.
Historical Reference	This is related to the concept of Wu Wei (effortless action) in Taoism. It can also be likened to a "flow" state. Basically, a good system architecture makes the user of the architecture more able to achieve these effortless forms of work without compromising the quality of the system.
Sources	Tao Te Ching Wu Wei, Andy Matuschak

Solve problems upstream
Principle 3

Tag	There comes a point where we need to stop pulling people out of the river. We need to go upstream and find out why they're falling in. (Desmond TuTu)
Descriptive	A good system architecture does not address only the symptoms of problems but addresses root causes.
Prescriptive	Do root cause analyses of any issues or symptoms and adjust the system to solve the problem at its cause rather than at its symptom.
Description	Problems are often more simply and better solved if dealt with earlier in the pipeline of the problem. This means examining the reasons an issue exists, moving up the causal chain to find the most fundamental cause of the issue, and solving the problem there. When we look upstream for a solution to a problem, it forces us to look at the whole system rather than at just a single symptom. This allows us to potentially bundle what would be multiple band-aids for individual symptoms into single systematic solutions. Any modifications to the system architecture should always aim to find the root cause of these issues and solve the problem at their source. Another way of describing this is by saying that the system architecture should address the cause, not the symptom, of discovered issues. If a system architecture only addresses known issues via band-aids directed towards the most visible symptoms, there will often be other manifestations of the root cause issue down the line. In more complex systems, System Dynamics analysis is one way that these types of complex system interactions can be discovered and resolved in a way that accounts for the difference between addressing the symptom and the cause. How can we best fix unemployment However, solving problems downstream is often easier because the symptoms may be clearer to see, and prescriptions faster to verify in a short period of time. For example, it's easier to see how seatbelts save lives than it is to see how lower speed limits save lives (crash tests vs large datasets).
Historical Reference	This is related to the Kaizen methodology used in Japanese work culture.
Sources	(principle conceived many years ago, original source unknown) Upstream, Dan Heath

Minimize sustaining energy

Principle 4

NOVEL

Tag	Do more with less. (Buckminster Fuller)
Descriptive	A good system architecture leverages existing energy flows to minimize the energy required to sustain the system.
Prescriptive	Find opportunities to convert explicit energy input into alternatives that leverage existing energy flows.
Description	As stated before, entropy is the tendency of a system to progress from order towards randomness. To fight this tendency towards disorder, a good system architecture should aim to reduce the energy required to sustain the system. Energy here means all types of energy, including physical energy, human effort, or systemic energy. Any form of "entropy fighting" is considered energy output, and a good system minimizes the amount of energy input required. An example of this would be two different systems, one that uses market forces to achieve its outcome versus one that uses government regulation. All else being equal, the one that uses market forces is a better architecture, because it uses the existing energy flow of the market to fuel its system rather than requiring the input of energy that is required by regulation.
Historical Reference	This idea is derived from the idea of exergonic and endergonic systems in chemistry. Exergonic systems are accompanied by the release of energy, and endergonic systems require an absorption of energy.
Sources	(principle conceived many years ago, original source unknown)

Eliminate hidden side effects

Principle 5

Tag	Every action has consequences, and it is the mark of a wise man to be aware of them. (Malcolm Gladwell)
Descriptive	A good system architecture has sub-systems that implement their intended functionality while minimizing any other outputs.
Prescriptive	Design sub-systems that do not produce hidden side effects.
Description	An architecture is made up of sub-systems, and these sub systems must be built to reduce side effects outside of their intended functionality. The more side effects are produced by sub-systems, the more unpredictable and unmaintainable a system becomes. Reducing side effects makes it easier to fight the tendency of a system towards entropy. One can think of side effects as "leaks" that may be harmless but have some probability of being harmful, especially at scale. Examples of side effects could be heat and sound in hardware systems, whereas in software it will mean any change to state caused by the function that is not the output of the function.
Historical Reference	While this principle may exist in different forms in different areas, my version is mainly derived from the core concepts in functional programming. Like in mathematical functions, functional programming is designed around functions that are idempotent and produce no side effects, making them flexible modules that are easier to work with conceptually.
Sources	(principle conceived many years ago, original source unknown) The Not-So-Scary Guide to Functional Programming, YLD

Assign sub-systems one responsibility

Principle 6

Tag	Every class should have a single responsibility: It should have a single purpose in the system, and there should be only one reason to change it. (Michael Feathers)
Descriptive	A good system architecture assigns only one functionality for each sub-system, with each sub-system being modular and replaceable.
Prescriptive	Limit all sub-systems to a single functionality.
Description	A good system is built off of many sub-systems. The modularity and flexibility of these sub-systems is vital to the long term viability and maintainability of the whole system. A good architecture is built such that these sub-systems are given a single responsibility that they are meant to fulfill. This single responsibility principle allows sub-systems to be optimally built to fulfill their one functionality. This avoids the problem of trying to create have single sub-systems that perform multiple functionalities but at a compromised capacity. This also makes it easier to change out sub-systems in the future. When a sub-system has only one responsibility, the calculations of whether a new sub-system should be used become much simpler, and the system as a whole avoids being overly reliant on one single sub-system.
Historical Reference	The Single Responsibility Principle is a concept that has existed in programming for many years.
Sources	I've vastly misunderstood the Single Responsibility Principle, Structure and Interpretation of Computer Programmers

Prefer simple over easy

Principle 7

Tag	Simplicity is prerequisite for reliability. (Edsger Dijkstra)
Descriptive	A good system architecture makes choices to reduce the number of steps and interfaces in the system over choices towards familiar or quick implementations.
Prescriptive	Choose to reduce dependencies and reduce the actions taken by a system whenever possible.
Description	The simplicity of an architecture is defined by the number of steps that are required to produce an output given an input. The simpler a system architecture is the more entropy resilient that architecture is. This is because there are fewer parts that can break. The easiness of an architecture is by how convenient and familiar an implementation is. Simplicity and easiness of a system are often correlated: simpler systems are generally easier. But a good architecture makes the decision to make simpler design choices when simplicity and easiness deviate from each other. The goal here is to reduce the burden of maintenance on a system by choosing an architecture that may not be easier, but is simpler. For example, just because a system has been built in the past a certain way, doesn't mean a different solution that reduces dependencies and steps should not be implemented.
Historical Reference	Multiple eastern philosophy give credence to the idea of simplicity, including Taoism and Buddhism.
Sources	An Intuitive Explanation of Solomonoff Induction Rationality, Eliezer Yudkowsky

Eliminate surprise

Principle 8

NOVEL

Tag	There are no surprising facts, only models that are surprised by facts. (Eliezer Yudkowsky)
Descriptive	A good system architecture embeds its prior probabilities of the world into its design, and builds in flexibility where the priors have large confidence intervals.
Prescriptive	Analyze your priors deeply and ensure the design of the system architecture account appropriately for the given prior probabilities.
Description	An architecture is, in some form, a model of the system that exists or will one day exist in real life. This model must be resilient to reality, and must have embedded in it priors that reflect the likelihood of events in reality. Priors are used in Bayesian probability as a basis for determining the likelihood of an event given no extra information. A model is surprised when reality exhibits a behavior that a model's priors gave low probability to occurring. This is something that an architecture must account for, and it must work to provide accurate priors for the real world. This means that if there is a 0.1% chance of a bearing failing, the architecture must be built to account for that frequency of failing. If there is significant uncertainty about the fail-rate of a given component, then the architecture must be flexible and resilient to the wide range of fail-rate possibilities.
Historical Reference	This concept is one that incorporates both Bayes' Theorem and ideas from the theory of communication (originated by Claude Shannon). This is also an important concept in epistemology.
Sources	Mathematical Theory of Communication, Claude Shannon Rationality, Eliezer Yudkowsky