The Frugal Architect

Simple laws for building cost-aware, sustainable, and modern architectures.

LAW I.

Make Cost a Non-functional Requirement.

A non-functional requirement specifies criteria that can be used to judge the operation of a system, rather than specific features or functions. Examples are accessibility, availability, scalability, security, portability, maintainability, and compliance. What often goes overlooked is cost.

Companies can fail because they don't consider cost at every stage of the business – from design to development to operation – or because they're not measuring it properly. The math is simple: if costs are higher than your revenue, your business is at risk.

By considering cost implications early and continuously, systems can be designed to balance features, time-to-market, and efficiency. Development can focus on maintaining lean and efficient code. And operations can fine-tune resource usage and spending to maximize profitability.

LAW II.

Systems that Last Align Cost to Business.

The durability of a system depends on how well its costs are aligned to the business model. When designing and building systems, we must consider the revenue sources and profit levers. It's important to find the dimension you're going to make money over, then make sure the architecture follows the money.

For example, in e-commerce, that dimension might be the number of orders. When orders go up, infrastructure and operation costs rise. And that's okay, because if your system is architected well, you can start to exploit economies of scale. What's important is that infrastructure costs have a measurable impact on the business.

As builders, we need to think about revenue – and use that knowledge to inform our choices. Because growth at all costs leads to a trail of destruction.

LAW III.

Architecting is a Series of Trade-offs.

In architecture, every decision comes with a trade-off. Cost, resilience, and performance are non-functional requirements that are often at tension with each other.

As the saying goes, "Everything fails, all the time." Being able to defend against failure means investing in resilience, but performance may pay the price.

It's crucial to find the right balance between your technical and business needs – to find the sweet spot that aligns with your risk tolerance and budget. Remember, frugality is about maximizing value, not just minimizing spend. And to do that, you need to determine what you're willing to pay for.

LAW IV.

Unobserved Systems Lead to Unknown Costs.

Without careful observation and measurement, the true costs of operating a system remain invisible. Like a utility meter tucked away in a basement, lack of visibility enables wasteful habits. Making meters more visible can profoundly shift behaviors.

Though observation requires investment, not implementing adequate monitoring is shortsighted. The adage warns, "If you can't measure it, you can't manage it." Tracking utilization, spending, errors, and more, is crucial for cost management.

When critical cost metrics are placed front-and-center before engineers and their business partners, more sustainable practices emerge organically. Ongoing inspection lets you spot excess spend and tune operations to trim expenses. The return on investment in observability typically far outweighs the expense.

Most importantly, keeping costs in the forefront encourages sustainable practices.

LAW V.

Cost Aware Architectures Implement Cost Controls.

The essence of frugal architecture is robust monitoring combined with the ability to optimize costs. Well-designed systems allow you to take action on opportunities for improvement. For this to work, decompose applications into tunable building blocks.

A common approach is tiering components by criticality. Tier 1 components are essential; optimize regardless of cost. Tier 2 components are important but can be temporarily scaled down without major impact. Tier 3 components are "nice-to-have"; make them low-cost and easily controlled.

Defining tiers enables trade-offs between cost and other requirements. Granular control over components optimizes both cost and experience. Infrastructure, languages, databases should all be tunable. Architect and build systems with revenue and profit in mind. Cost optimization must be measurable and tied to business impact.

LAW VI.

Cost Optimization is Incremental.

The pursuit of cost efficiency is an ongoing journey. Even after deployment, we must revisit systems to incrementally improve optimization. The key is continually questioning and diving deeper. Programming languages provide profiling tools to analyze code performance, and while these require setup and expertise, they enable granular analyses that can lead to changes that shave milliseconds. What may seem like small optimizations accumulate into large savings at scale.

In operations, most time is spent running existing systems. There are opportunities to profile resource usage and identify waste reduction. At Amazon, we continuously monitor services in production to understand patterns and trim inefficiencies. Frugality takes persistence – by incrementally reducing latencies and infrastructure costs, we can optimize the cost to serve.

There is always room for improvement… if we keep looking. The savings we reap today fund innovation for tomorrow.

LAW VII.

Unchallenged Success Leads to Assumptions.

When software teams achieve significant success without facing major failures or roadblocks, complacency can set in. There is a dangerous tendency to become overconfident in the methods, tools, and practices that led to those wins.

Software teams often fall into the trap of assuming their current technologies, architectures, or languages will always be the best choice, simply because they have worked in the past. This can create a false sense of security that discourages questioning the status quo or exploring new options which could be more efficient, cost-effective, or scalable.

You see this frequently when it comes to programming languages. “We’re a Java shop” is a phrase I’ve heard too often – and it can stifle innovation. Unchallenged success breeds complacency through assumptions. We must always look for ways to question, optimize and improve.

As Grace Hopper famously stated, one of the most dangerous phrases in the English language is: “We’ve always done it this way.”