Protect Your Shed

April 8, 2026 · Dylan Butler

Constructing a skyscraper is a massive undertaking. You need architectural blueprints, council permits, and safety audits before the first piece of steel is even ordered. It requires hundreds of people coordinating over months or years. You can't just throw up some drywall and hope the building holds weight.

Then there is the backyard shed. No blueprints, no permits, no audits. You just grab some timber, a saw, and start hammering. It might be a little drafty, and the roof might leak if it rains too hard, but you built it yourself in a single weekend.

For the last six years, my life as an engineer was split between these two modes. By day, I was building banking systems at enterprise scale. By night, I was in the shed, building whatever I felt like. Side projects that sometimes went somewhere and sometimes didn't.

It is easy to view these as two separate lives: the work you do for a paycheck and the work you do for fun. But looking back on this chapter of my career, I've realised something fundamental. The enterprise work taught me how to engineer at scale, but it was the personal projects that kept me an engineer.

I've always told younger developers that maintaining side projects will do more for your career than any amount of interview prep and LeetCode will.

Learning the physics of scale

Early on, what stands out is how much of the work isn't actually writing code. There are design documents, test plans, architecture reviews. It can feel like the actual building part is a fraction of the job.

But that surrounding work is what makes the building possible at scale. When you're processing the volume of transactions a major bank handles, you can't skip the design phase or cut corners on testing. Each of those steps exists because someone before you learned the hard way what happens without it.

Working in that environment gives you access to unattainable scale. You get to work with tools like Cloud Spanner, a globally distributed, strongly consistent database that you simply cannot simulate on a laptop. You learn defensive design. You start thinking about failure modes before you think about features.

But that scale comes with a cost: rigidity. You are a single worker on a massive site. You don't often get to choose the materials, and you rarely get to experiment with the foundation.

Taking the blueprint home

The shed is where you take the blueprints you learned on the job and actually get to play with them.

In the early days, my personal projects were messy. Architecture was an afterthought if it was a thought at all. Classic shed behaviour. But over time, the patterns from work started bleeding in naturally.

You spend enough time designing systems that need to handle failure gracefully, and you start doing it on autopilot. The homelab is probably the best example of this. What started as a single container on a single machine turned into a managed cluster with automated deployments and infrastructure defined in code.

That's taking the structural discipline from the skyscraper and applying it to a space where I had total freedom. The personal projects stopped falling over. They were still built fast and on my own terms, but they were anchored. The enterprise taught me the rules of structural integrity, but the shed gave me a place to actually be the architect.

The freedom to break things

When you're building for yourself, the cost of a bad decision is a wasted evening. At work, choosing the wrong approach affects real teams and real customers.

That rapid feedback loop is what makes the shed so valuable. You are the developer, the reviewer, and the user. You can tear something down and rebuild it just to see how it feels.

I built a Game Boy Advance emulator in Go not because the world needed one, but because I wanted to understand how hardware works at that level. I've stood up services using tools I'd never touch at work just to understand their tradeoffs. You can try a tool you've never used before without writing a proposal for it.

Most of these experiments don't turn into startup ideas, but they all leave something behind. A new pattern, a lesson in what not to do, a broader sense of what's out there.

More than anything, the shed is where the curiosity stays alive. Enterprise work is highly valuable, but it can wear you down. Sprints blend together, the ticket queue never shrinks, and the problems start feeling repetitive. Personal projects are where you go to remember that building software is actually fun.

Bringing the hammer to the skyscraper

Earlier in my career, I was new to containerisation and cloud infrastructure, and the learning curve at work was steep. But because I was standing up containerised systems and running them on GCP at home on my own time, the concepts landed faster. I was getting reps in from both sides.

This pattern repeated throughout my career. You try something in the shed on a weekend because you're curious. You learn the tradeoffs, the rough edges, the things the documentation doesn't tell you. Then months later, when the team at work is evaluating that same tool or approach, you're not starting from zero.

Because you've already broken things in your own environment, already evaluated tools on your own, and already felt the pain points, you can show up at work and make informed calls instead of guesses.

Protect your shed

The trap of software engineering is thinking that your day job is the entirety of your craft.

The engineer who only builds skyscrapers eventually burns out. The problems become repetitive, the process becomes suffocating, and the creative spark starts to dim. You stop building things because you want to, and start building them because the business says so. You lose your edge.

Protect your personal projects at all costs. It is where your curiosity lives, where you experiment, and where you define yourself as a builder rather than just an employee. The enterprise will teach you how to write code that survives, but the shed is what ensures you actually still want to write it.