

Building a System

Walid

One of the many things I discovered that really matters as an engineering student; and something I was (like many of my peers who went through the same academic path) left behind on; is systems: how we can build them, how to design them (fun fact: for a long period I thought design was just drawing), and how to set up their architecture.

I started dabbling with these ideas during my internship when I had to build a system linked to a database (which was not giving much output at the time) in order to transform those data points into KPIs that would render many operational aspects of the plant. The tool used for extracting data from the points (machines) was one of the subcomponents that would be used by the system, but it was fundamentally broken, so I had to work on adapting everything to fit the system. I started by laying out six figures generated by matplotlib. I was happy with the result until, after long discussions with staff and operators, I discovered that graphs are not liked by most people; they remind them of school, which is not good (they automatically make them subconsciously stop paying attention). That's when I knew design matters a lot.

Long story short, I built an automated web page where we could see the KPIs after they traveled through a pipeline that involves pulling the data points from the database, normalizing them, doing the calculations, and then representing them in an efficient, color-coded ways based on thresholds.

Even though I finished my internship, I'm still not satisfied with the work because I feel I did not defend my case properly. I knew that requiring the manager to run a Python command to get the HTTP page wasn't user-friendly, and my prediction proved correct; you can guess how many times they ran it over the last two months (I would say three times at best).

But since then, my ideas about building and laying the ground for a system have shifted from 0 to 0.4 (once a Walid who didn't see solutions through a systematic lens, now a Walid who leverages the architecture and the robustness of a system; 1 would be senior Walid who does the design and architecture in his mind), mostly inspired by N. Taleb's ideas on antifragile systems and P. Thiel on startup building. You might see it as irrelevant, but I have my own way of combining them with other ideas.

When setting up a system, I should start by determining its components, since those components generate the state of the whole system. From an architectural point of view, the system must be centralized and decentralized; it must be at the center of the powers each component can provide and not affected by the harm one component can do.

I like to see the first idea in the human body: simply, if I take care of my body's components, the whole body gets upgraded. The second part is solved by redundancies (mostly spatial and aeronautical); when one component gets stuck or triggers a reboot, it gets replaced instantly. But it's not the replacement that matters. Yes, it got replaced during a flight so we don't have a disaster; that's wonderful; but for me, getting that component down and knowing what went wrong is the essential part, because it will shrink the probability of other accidents by many folds.

The evolution of the system is linear; hence its components (or vice versa); and it can't be unidirectional or have the same direction. Component X_1 , responsible for storing data, can be changed and is also part of another pseudo-system that can dictate the evolutions and bring new stressors to the frame. To elaborate more on my X_1 : the database can be used by maintenance to work on preventive maintenance via ML models, and those models will need non-trivial info,

so we can add or merge columns. Also, if the shifts get intense and the flow of data gets bigger, the whole pipeline will need an update.

It may seem complex and entropy-savage, but these two come as results of bad architecture. So to make the idea clear: if the plant gets bigger and my X_1 has many new stressors (information), we can redirect the flow of thrust to other X_i so it can hold more refined output. By upgrading those components, the pipeline as a whole will become antifragile, and it can pave the way for us to reshuffle the components.

The nasty part is trade-offs: since the components are not evolving in the same directions, there will be trade-offs.