Planned Obsolescence in Software

Refactoring is always the right choice. Or maybe rewriting is the right choice. One of them.

A purist in me would alike to think that refactoring an aging system is always possible and the cheaper option for the business. I'm of the mind that it's *probably* true most of the time. If software was just built in certain ways! (waves hands)

I tend to like Restoration Software. Scraping the paint, sanding down the roughness, applying stain. Weeding a garden. Rebuilding an engine. Making an old thing new and better *without breaking it*. Consequently, I've chase those types of projects and have done a lot of it. I can claim competency and next decade, expertise.

I say that only to brag a little and to point out a bias of mine. I oddly prefer the brownfield project to greener pastures. I say oddly because this is one of those areas where the industry has agreed I am wrong.

The major pattern I've seen is slowly letting older systems go out of style until you replace them. Maintenance of systems is a large part of software building, of course. But it's not a sweet, tender tuning of a purring engine. It's dragging a trash can full of water around until you find enough duct tape to stop up one of the many leaks. Then after you stopped up one, someone DMs you "that trash can you built last year is on fire."

I make a sincere effort to build a less leaky, incombustible trash can. I get feedback that people like working on my systems, that they are easy to extend to meet needs of the company after I have moved on. Even so, out of the 20 trash cans of 'mine' (read: the company) out there, at least 10-15 of them are leaking and on some sort of super fire. Those are the ones I learned by making mistakes.

And the bet would be they've all been replaced, or fallen out of importance. I'd bet on this because this is how it's always been.

The industry seems to have decided that rewriting is per of the software lifecycle. Microservices was the high water mark of that software fatalism. A technical implementation of the idea that code is meant to die, to be rewritten, to sleep(inf), perchance to dream.

Ay- we maybe seeing the wave roll back. A move towards services, yes, but with an awareness of the complexity the network hops bring.

I concede I'm being obstinate.

My experience with software tells me there's a *nature* to it. There's a material of it. There are ways it works and you can bend those. The shape of software is slippery and hard to define. These "rules" (??) feel a little bit like math; there are logical conclusions. They are weird and rigid and ethereal, but they are there and *you better get used to it*.

And one of those rules is sometimes systems die. The rewrite will make more sense. It will be cheaper. The bugs a rewrite will inevitably incur (my estimate at some 10x more than a refactor- to eee is human) are preferable to the extra cost and effort of a refactor (also at approx. 10x)

I'm speaking in generalities; there are a million variables: business stage/mode/model, fault tolerance of domain, language, tools and technical design, etc. but my whole point here is that I'm wrong, I guess. While a successful rewrite still requires extra effort and work to confirm results and minimize bugs, it's a logical choice.

My critique and hypothesis is that the industry favors rewrites unwisely. Uphill analysis, downhill creation bias. Career climbers rallying business leads around new products or new technologies. The promise of the blank page. The promise that this time, we will get it right.

I don't want to stop chasing a dream of building but-it-for-life, endurable software. For me, refactoring makes as much pragmatic sense as darning a sock instead of throwing it out.