The Domain Engineer

Why the rarest person in health IT is the one who carries two worlds at once, and why AI makes that person more valuable, not less.

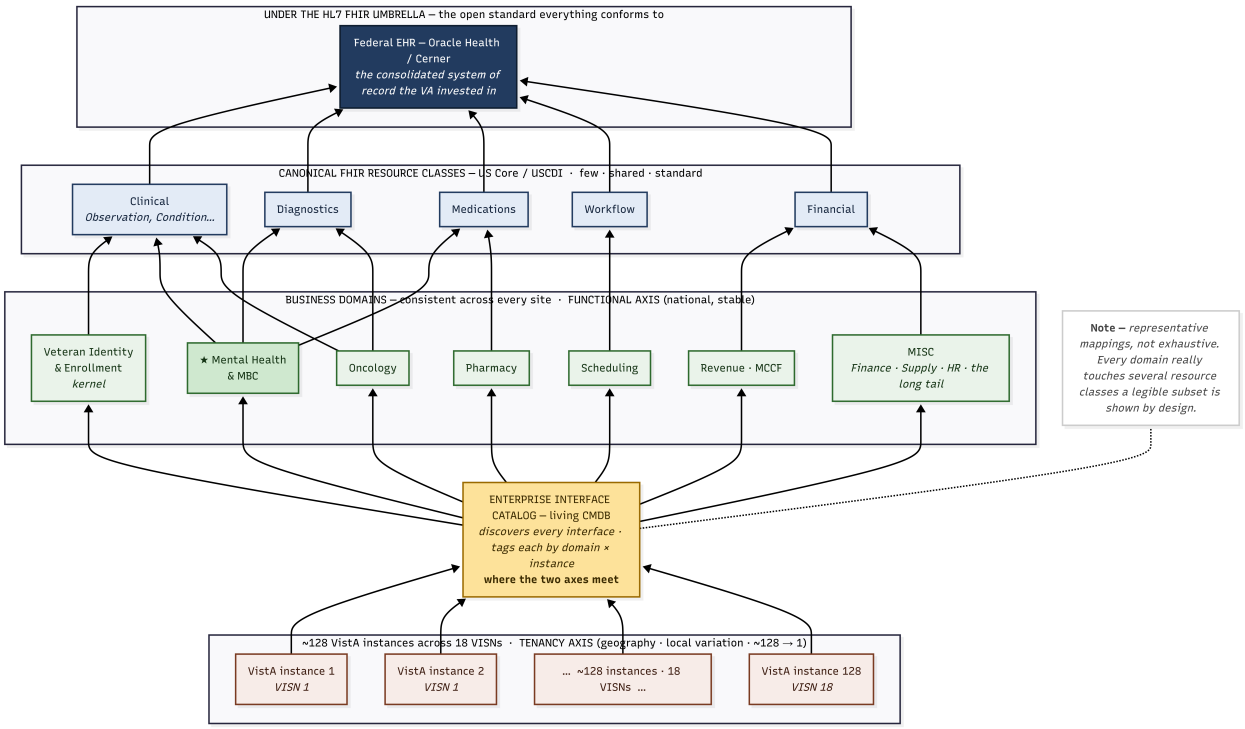
It is late, and I am reading a government RFI my company has no business being this excited about. Not because the problem is beyond me. Because this is department-wide work, the kind only a short list of pre-designated primes, the ones already holding seats on the vehicle, are even allowed to propose on. We are simply years of building solid past performance away from a seat of our own. And right now the VA is doing what it does at this stage, asking the industry for answers, and these are answers I have.

It is the usual wall. Acronyms stacked on acronyms, a scope that could mean everything or nothing, the flat procurement voice that files a life-or-death system under "modernization requirements." Most people read a few pages, decide it is somebody else's problem, and close the tab. I have done exactly that plenty of times.

This one will not let me go. Because underneath the standards language and the interface counts, I can feel a shape starting to surface. Not the words on the page. The thing the words are circling without quite saying. And before I have really decided to, I am sketching.

Two axes. One runs across the functional work of care, the business domains that stay the same no matter where you stand: identity and enrollment, pharmacy, scheduling, mental health, oncology, the long tail of finance and supply. National, stable, the same in Maine as in Arizona. The other axis runs the other way, down through geography, roughly a hundred and thirty local systems, each with its own history and its own quiet variations, all of it supposed to roll up into one. And the interesting part, the part that makes the whole thing tractable instead of terrifying, is not either axis on its own. It is the single place where they cross. One catalog that knows every interface by what it does and where it lives, sitting at the seam, letting the standard sit calmly on top.

By the time I look up it is very late and there is a diagram on the page that was not there an hour ago.



What was under the wall of acronyms: two axes. The functional business domains, national and stable, and the roughly 128 local VistA instances with all their geography and local variation, meeting at one living interface catalog, everything conforming up through FHIR to the consolidated federal EHR.

I have no superpowers, but what I do have is a different view on most things than my peers do. Not better, just different. I sit at several intersections of things that usually do not overlap: engineer and soldier, white collar and blue collar. A person who builds incredibly complex systems and a person who cannot stop feeling for the people caught inside them. I often have curiosity well past the point of usefulness. I have played no real part in most of what assembled me this way, but I have decided to make the most of it, and of the short time I have here.

What those intersections buy is coverage. Years in the behavioral-health domain taught me how care actually moves, what a number means and does not mean, where people fall through. The other side of it I learned somewhere else entirely, in other industries with their own messes: modernization, the unglamorous craft of dragging decades-old systems forward without dropping what they carry. Interoperability is not a separate discipline there. It is baked in. You rarely get to flip a switch from old to new, so the two have to run side by side and work as one system, with no gaps where something falls through and no overlap where it gets done twice. That flavor of interoperability is just the cost of modernizing anything real. Healthcare means more by the word, and I will not pretend my other-industry work makes me fluent in all of it. But the discipline underneath, making the old and the new work as one without dropping anyone, is the part that travels.

Most days those live in separate rooms in my head. Reading that RFI, they were both switched on at once, and so was everything else I am. When enough of that is loaded at the same time, the

pattern is simply there, available, obvious in a way it is not to someone standing in only one of those worlds.

That is the whole subject of this piece. Not the diagram. The kind of person the diagram comes out of.

The best label I have for this is: domain engineer. We all know frontend, backend, and fullstack engineering roles. We know the product owner role too. The product owner carries the domain. The engineers carry the build. But what about that rarer one in the middle, held by someone with enough of a real field to know what actually matters in it, and enough engineering to build something that respects that? The domain engineer carries both, and needs no translator between them. The person who can read the wall of acronyms and see the product underneath, and the people it serves.

None of this is new, and I do not want to pretend it is. Domain-Driven Design has been a thing for years. It insists that software be shaped around the language and the boundaries of the business it serves, that a "context" is a real thing with edges you cross at your peril. These days DDD mostly gets talked about as an architecture pattern: carve the system along those boundaries, let each service own its own slice of the logic and the data, and let everything between them settle to eventual consistency. That is related to what I am after, but it is not quite the same. What lasts, underneath the patterns, is the original insight: context is king. It was true when they said it, and AI has only made it more true, by giving the rest of us a fashionable reason to finally listen.

Here is the smallest example I know of why it matters, and it is one you can see at both layers at once.

In a modern health record, a depression screening score and a cancer patient's lab value are neighbors. Both are the same kind of thing to the machine, the same resource class, the same code-and-value shape, both perfectly valid against the same standard. At the data layer they sit side by side, indistinguishable, and a pipeline that only sees the shape will treat them as the same. It is not wrong to. It is just blind to everything that matters.

Because how each one got there could not be more different. One is a person answering nine questions about their own mind, subjective by design, meaningful mostly as a trend over time. The other is a machine measuring a body, ordered by a physician, often actionable the moment it lands. Same shape. Opposite provenance, opposite reliability, a different clock, a different person who is supposed to do something about it. Flatten them into "just another observation" and you will ship something that passes every conformance check and still misleads the clinician reading it, an alert that fires wrong, a trend line that lies, and eventually a model trained to trust a feeling like a fact.

The domain engineer is the one who looks at those two identical-looking things and says, quietly, these are not the same, and if we do not honor where each of them came from we are going to hurt someone with software that was technically correct the whole time.

The machine is very good at the part I am not

For a while the story everyone told about AI and people like me was a threat. The machine writes the code now, so what is the engineer even for. I understand the fear. From where I sit it has the shape exactly backwards.

Here is what I actually do on a good day. I know the context. I know that a screening score is not a lab value, that a hundred and thirty local systems each carry their own history, that one field in one message can mean something entirely different from the identical field a system over. That knowing is the slow, expensive part, the part that took years of living in more than one world to earn. And once I have it, the rest, the digestion, the volume, the thousand careful translations, a model can increasingly do. A lot of the time that is all execution needs. Someone to hold the distinctions, and something tireless to carry them out.

So the machine did not take the valuable part. It took the part I was never precious about. What is left, the part it cannot do on its own, is decide what matters. A model will happily flatten the screening score and the lab value into the same shape, because nothing in the data tells it not to. It needs someone to hand it the distinction, to say these are different, here is why, and here is where they must never blur. Context is not the thing AI replaces. Context is the thing AI runs on. Feed it ungoverned, un-conformed, undistinguished data and it will produce fluent, confident, wrong answers at a scale no human could ever match.

Which is why the domain engineer gets more valuable in this, not less. The scarce input, judgment about a real field, is exactly the input the model cannot generate for itself. Pair that judgment with a tireless executor and one person carrying two worlds can do the work that used to take a room.

The posture that makes it work is a specific one, and I think it is the whole trick. Humble and hungry at the same time. Humble enough to let the machine do what it is plainly better at, to not confuse being needed with doing every keystroke, to admit that most of what I build stands on standards and tools and people who are sharper than me in their own corner. Hungry enough to keep feeding it the right context, to keep learning the domain deeper than the model ever will, to keep reaching for the next thing. Humble about the how. Hungry about the what.

That is the version of this work worth wanting. Not a person who resents the tool, and not a person who hides behind it. A person who knows exactly which part is theirs to carry.

The system that lasted is not the enemy

There is a reflex in this work I want to name, because it is the opposite of everything I have been describing. Someone looks at an old system, the one that has quietly run the core of an operation for decades, and sees only an eyesore. Ugly, ancient, in the way. The instinct is to dynamite it and start clean.

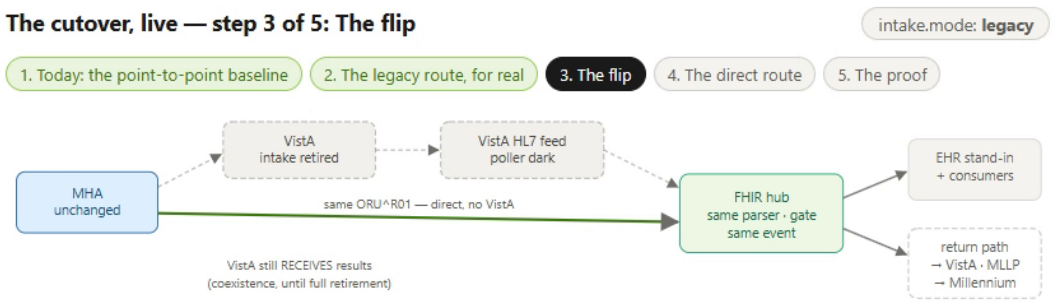
That reflex has it exactly backward. A system that ran the heart of something for forty years without falling over is not junk. It is proof. It survived contact with reality, every strange edge case

and compliance rule and one-off exception, and all of it is still in there, load-bearing, battle-tested in a way you could never re-derive from a blank page. That is not debt. That is gold, refined intent that endured. The people who walk past it seeing an eyesore are walking past a goldmine.

But it is a mine, not a bar sitting on a shelf. The gold is embedded in rock, tangled up with dead code and obsolete rules and workarounds nobody remembers the reason for. And the shaft itself is old and unmapped, the original diggers retiring one by one, carrying the map out in their heads. So the danger and the value are the same fact. That is the whole tension of it.

Disrespect it and you get a cave-in. Lift and shift it, assume the weird branch is garbage, delete the thing that turns out to be the only code handling a real case, and you find out in production, which is to say you find out when it has already hurt someone. Respect it and you walk out with the gold and everyone still breathing. That means assuming every strange thing is there for a reason until you have proven otherwise. Shoring up as you go instead of blasting. Following the veins carefully.

That is the domain engineer's stance toward the past, and it is the same stance as everything else here. Respect where it came from. The legacy system is not the obstacle in front of the work. A lot of the time it is the work, and it is worth doing right.



Respecting the old shafts, in practice. At the flip, the feed stops running through the legacy intake and sends the same event straight down the new route. But the old system keeps receiving results on a return path, coexisting until it is fully and safely retired. No big bang, no gaps, nothing dropped.

The way of thinking I want to spread

You cannot hire this off a resume checklist. There is no line item for carries more than one world at once, no certification in sees the people under the acronyms. The usual machinery of who gets to build important things looks right past it, because it is hunting for the specialist or the generalist, and this is neither.

Here is the part that matters, the reason I am writing any of this down. It is not a fixed trait. It is not something you were handed at birth or missed. It is a way of thinking, and ways of thinking spread. You can decide to stay curious across a boundary instead of stopping at the edge of your lane.

You can decide to learn a domain like the people in it are counting on you, because somewhere they are. You can decide to respect where a piece of data came from instead of flattening it to fit the pipe. You can let the machine do the digesting and keep the judgment for yourself. None of that requires my particular strange assembly of a life. It requires the choice to work that way.

So I am not trying to collect these people. I am trying to spread the habit, or at least make it more common than it is. If more of us read the hard, boring document and feel the shape of the people underneath it, better things get built, and fewer of them fail the person they were meant to serve.

If you are already that person, or you want to be, we are working on the same thing whether we ever meet or not. And if you build, buy, regulate, or live inside health systems, and any of this landed for you, I would like to talk.

Standfast Systems. Always forward.