Software Architecture Guide (martinfowler.com)

803 points by rspivak 71 days ago | hide | past | web | favorite | 288 comments

Ozzie_osman 71 days ago [-]

Lots of hate in this thread.. People asking for proof: what sort of evidence would convince you? I take these blog posts, apply it to my experience, and take what I think makes sense. Sometimes, an idea will solve an obvious pain I've had. Sometimes, an idea will show me a pain I didn't know I have. Sometimes I disagree with the idea because it won't work for me. That's fine. I'm still much better off thanks to this literature, and I would be worse off if it all had to be evidence-based.

If you think you can write better based on your experience, please go for it. I'd love to have more literature to read, analyze, and learn from. If you can't (pressed for time, etc), and only have time to post in HN comment threads, then at least discuss concretely which parts you disagree with so we can learn from that too (I'm serious, not being facetious). If you can't even do THAT, then sorry, all your valuable experience is way less useful than someone like Martin Fowler, because I'm not learning anything from you.

Having a good understanding of how to architect code is hard. Being able to write about that in a way that's useful for others is even harder, and is a skill of its own.

So personally, I thank Martin & Co for all their writing.

jonbronson 70 days ago [-]

Thank you for writing this. As an engineer reading these style of articles, I've witnessed two common reactions: Either the content resonates, and our eyes widen at the site of a simplified and intuitive explanation for phenomenon that we've somehow already observed, or... it doesn't. And maybe that's the source of much of the negativity we're seeing. These engineers haven't worked in domains, projects, or organizations where they could encounter how these lessons play out, and so they're dubious about the whole thing. One reaction to being in that position is to downplay the author and complain about lack of 'evidence'. The other, is to put down our egos, and consider that maybe we're just not there yet to appreciate what the author is trying to tell us.

blub 70 days ago [-]

That's not at all the source of my negativity. Instead, I am confronted in my work with developers of different experience levels which justify their design and coding decisions based on what e.g. Robert Martin or Martin Fowler wrote in a book or blog.

Some of these ideas, like "code should read like prose" or the obsession with unit testing are causing problems, such as making the code harder to follow or neglecting other kinds of testing that are more effective.

I'm not going to claim that everything these authors say is false, but it's high time we say: PoC or GTFO - show us the systems where these rules are applied and let us judge if they're worth anything, don't tell us nice stories.

tempguy9999 70 days ago [-]

> unit testing [...] or neglecting other kinds of testing that are more effective

Could you elaborate? (happens I come from a background where there is no kind of systematic testing, which is a bit of a downer)

And 'PoC' = proof of concept?

> which justify their design and coding decisions based on what <authority> wrote in a book or blog.

Happened to me. We had an SQL-based product that ran like a dog. Much faster by the time I'd finished it (I was the local guru), but on the next iteration I found the lead programmer had ripped out the big complex SQL statements it generated, and replaced each with many simpler queries. I asked WTF he'd done that without telling anybody and he justified it by pointing to a blog article recommending that for speed.

The article ended up 'trust me, it will be faster'. Oh yes, it ran faster because the query optimiser (very cpu expensive) now had very little work to do - so in ran like a rocket when the lead programmer (ie. *one single user*) tested it in his laptop. But it would scale like a dog because the optimiser had been sidestepped!

In addition, lots of little queries picking data from here and there would allow inconsistent views of the data because he didn't know about transaction isolation levels.

Thanks, idiot blog author!

chii 69 days ago [-]

i dont think having a blog outlining random advice is the source of the issue - it's the programmer who chose to take advice without first measuring and/or critically evaluating it.

Even if martin fowler and co didn't write this, soembody else would've, and the same sort of mistakes would continue to happen. The root cause, as always, is incompetence.

tempguy9999 69 days ago [-]

If people don't know much then they are not in a position to discriminate. It's the unknown unknowns. In the case I gave I'd reasonably agree that the lead programmer should have sat down with a stack of MSSQL books and done his homework.

But even more so, the prat of a blog author, a well-known blogger in MSSQL at the time, should have done his homework and not published dangerous crap. Proclaim yourself an authority, you'd better be one! I've other examples of this too.

But sometimes the end-users just can't know. I'll skip the details but due to overload at work I couldn't help our support staff properly, so they did a web search and grabbed a snippet which did what they needed. Well done for initiative, lads.

It also silently disabled full trans logging on our clients production DBs. Not so good. Support staff couldn't know, they did their best, blame lies with idiot blogger 'expert'. Again.

blub 69 days ago [-]

I was referring to system and integration tests. If a system has any kind of user interaction, user testing is essential.

ChicagoDave 70 days ago [-]

The articles on Fowler's blog are written by well-established senior developers/architects with years if not decades of experience.

There's no evidence that Fowler, et al, are saying there is One True Way and he even eludes to the fact that any such proposal makes him uncomfortable.

I also have decades of experience and would add my observations:

Leadership matters. If you're a part of team without authority or with weak a vision, you suffer at the whims of your leadership. I find this to be the first point of failure. Without a strong vision, enterprise software will develop code debt instantly and it will grow exponentially. Enabling a development team should be a priority.

Vision and Strategy A significant effort should be made to reason through objectives and put guardrails on those objectives. Are you re-platforming? Are you only rebuilding one boundary? Are you migrating to the public cloud and leveraging IaaS and PaaS? How brittle are your existing systems? Can your existing systems support the expected growth of the business?

Someone needs to consider these things before engaging with system change, develop a plan, socialize the plan with everyone, then develop execution strategies to enable the plan to succeed. This has less to do with code, but drives coding standards.

If you have complex systems, understanding Domain-Driven Design is a critical aspect to improvement.

If you have a “big ball of mud”, pulling it apart and rebuilding those parts according to the business is critical and can’t be done by “winging it”.

Opinionated vs Unopinionated Deciding how high your development guard rails need to be and managing them is an art, but it still needs to be explicitly managed. In an enterprise, opinionated is probably better. In a startup, the opposite may be true. Someone or some team needs to think about this and talk about it.

Communication Everyone, from Product Owners to Testers to QA needs to be able to model, whiteboard, and discuss problems, scenarios, and agree on solutions. Throwing work over walls will destroy the expected results.

goto11 70 days ago [-]

I think there is some tribalism at play. Fowler and friends are writing about enterprise architecture. HN is probably more oriented towards startups which haven't yet reached that kind of challenges and constraints (and in most cases never will), and who feel superior to the dinosaurs. And then there is is the "computer scientists" who think software development should be based on proofs rather than experience and results.

sidlls 70 days ago [-]

And then there are people like me. I've worked in big enterprise software departments (a huge homeowners insurance tech company, big banking, government contractors), "companies" of three people including myself, and some in between. Currently I'm at a 500-ish person company that is not quite enterprise but not really a start-up.

My experience is that "Enterprise Architecture" creates as many problems for the business as it solves for technology practitioners attempting to solve problems in a huge battle royale of competing departments. I think it often helps cement and provide feedback loops for the *causes* of the problems seen in large enterprise systems.

goto11 70 days ago [-]

This is where I get confused. Are you criticizing a particular architecture, or architecture in general? Is it even possible to have a system without an architecture?

james_s_tayler 69 days ago [-]

The Winchester Mystery House does exist in software somewhere.

JeffGrigg 69 days ago [-]

Even the Winchester Mystery House has architecture. Pretty good architecture. It's still standing, in spite of earthquakes. She bought it in 1884.

Now it has some fairly odd "features" based on her very eccentric "requirements." Doors that open into walls. Stairs that "go nowhere." *Very expensive* windows that look out at brick walls.

But given her odd sense of "requirements," it's the house she wanted!

blub 69 days ago [-]

The word enterprise is not included in the article, which makes it a generic article about architecture.

Experience and results are better then nothing. Where are the author's results?

Based on which project experience was the article authored? Without knowing such things, we can't distinguish what kind of outcome will the proposed ideas result in. After all, good results can be good on many dimensions.

hyperpallium 71 days ago [-]

Yes. And Computer Science is not a science, and Software Engineering is not an engineering.

Guides to it are more like guides to writing. *Strunk and White*, where is your empirical data? What's your control? Where are your PDE's?

Some also have aspects of business productivity books, with well-known common-sense ideas, like *In Search of Excellence* (which made up case studies, and a co-author said what's the problem?)

Even Fred Brooks' *Mythical Man-Month* - informed by his leadership of the massive bet-the-company "360" project at IBM - lacks data. It's still great though (in, uh... my subjective opinion, I guess).

eitland 70 days ago [flagged] [-]

> Even Fred Brooks' Mythical Man-Month - informed by his leadership of the massive bet-the-company "360" project at IBM - lacks data. It's still great though (in, uh... my subjective opinion, I guess).

I have just read that book.

It is the computer book with most tables pr page of any I have seen.

How you come to this conclusion is beyond me.

hyperpallium 70 days ago [-]

The "data" in this context is empirical data in support of the hypothesis that the man-month is mythical (aka adding people to a late project makes it later). You also need a control.

Similar for other hypotheses, such as a "programming product" and "programming systems" take three times longer than a "program" (written by e.g. a garage duo).

BTW Your "beyond me" was unnecessary.
https://news.ycombinator.com/newsguidelines.html

hyperpallium 69 days ago [-]

Thought I'd better check this. I went through the book's eponymous chapter 2 *The Mythical Man-Month* (p.14, 1975 ed), and it has not a single table; not a single datum.

It has many graphs, without data points, and the text and their smoothness indicate they are graphs of functions, not data.

I have a disturbing feeling that the downvoters in this thread simply don't know what it means to validate a mathematical model with data.

BTW It's not a criticism of the field that it's not yet an engineering. It's just where it is at the moment.

blub 69 days ago [-]

A philosophical book of essays about software from 1975 is not relevant for a set of recommendations on architecture 44 years later.

You are correct that chapter two lacks supporting evidence. And that's a problem for the book, it doesn't mean it's ok to omit data everywhere now.

JeffGrigg 69 days ago [-]

I think it's quite clear from context that all graphs in Mythical Man Month are based on formulas and theory, not on double-blind experiments or empirical measurements.

Still, I often find it quite useful advice even for today's projects, where people commonly assume that they can put a late project back on track by adding people to it, and that by doubling the staff, the product will be produced twice as fast.

eitland 69 days ago [-]

I seems to me that you and hyperpallium agree.

I still hold that

1.) That book is still one of the most relevant books I've seen lately (which might be scaring, I haven't made up my mind yet)

2.) While he might not have (or at least not include) data, he argues hos case well, and while I wouldn't bet money on experiments to follow his smooth curve I would be willing to bet (a small amount of) money that for non-trivial software his ideas would still result in better estimates than one that accepts "man-months" as a unit of progress.

And for those that haven't read the book yet it might be worth noting that while some people might find the book quotable more or less from cover to cover there is also a number of observations to be made that takes a full reading to realize. Here is one I'd like to point out:

Some quotes from the book might leave one thinking that he says programming cannot be scaled. I'd say the opposite is true, he just argues that it must be done in a thoughtful way instead of by mindlessly adding people to the project.

blub 69 days ago [-]

I don't have an opinion on the book, I read it too long ago and just skimmed chapter 2 now. I don't think it's comparable with or relevant for this blog post.

hyperpallium's original message, the one that started this thread is a feeble attempt at legitimizing the blog post by comparing it to a bunch of unrelated and generally appreciated things.

hyperpallium 69 days ago [+3]

streetcat1 71 days ago [-]

Software engineering is engineering in the same sense the mechanical engineering is engineering.

The only difference is that with software your materials are descriptions. I.e. you describe something, give to the computer and the computer turn itself into this something .

In the end, there is a physical entity in the world (the hardware), which turned itself into a unique machine based on your description (for example an ATM).

While in mechanical engineering, you materials are physical, which might make it easier to explain.

hyperpallium 71 days ago [-]

That is a valid argument if your concept of a mechanical engineer includes anyone who designs a machine; for example improvising levers, pulleys, gears to achieve some end.

I guess it is all "engineering" in a casual, tinkering sense. But a professional engineer, with a Mech Eng degree who uses mathematical models of stress and strain etc, who did lots of tinkering as a kid, might not agree.

ende 70 days ago [-]

Again with the over emphasis on “state sanctioned recognition”.

TeMPOraL 70 days ago [-]

So will you call anyone that tells people what chemicals to ingest when "a doctor"? Will you call anyone on the Internet commenting on legal issues "a lawyer"? Or would you prefer people emphasize the meaning of "state sanctioned recognition" in these cases?

Engineering is like law and medicine, in that it requires high skill and can hurt a lot of people if done badly. Not every country has engineering licensing, but I find myself increasingly agreeing with those that do. I feel calling software development engineering is mostly false advertising right now, only trying to capitalize on (and devalue) the status that engineering has in society.

Ozzie_osman 71 days ago [-]

I disagree. Engineering disciplines vary in how far removed they are from the underlying physics and how malleable they are. Building a bridge (or civil engineering) is very connected to the underlying physics. A set of equations give you the answers. It's also not malleable. You design it once, then build it, putting in some margins of safety to account for any variances in underlying material or potential usage, then it's pretty much not malleable.

Mechanical engineering is a little more removed. You have moving parts, interconnected pieces, wear and tear, etc. Now you have to lean heavily on preventative maintenance, ability to replace worn parts, etc.

Hardware is more removed. Software is even more removed. The laws of physics are there, but there are so many man-made layers of abstractions on top of it that "good architecture" is a lot less like what you'd expect in a typical engineering discipline.

Not saying any is easier than the other, just that they are very different disciplines.

holy_city 70 days ago [-]

I strongly disagree.

Engineering is the study and practice of problem analysis and design synthesis. Whether that means using abstractions of the physical world or how data is structured and processed is unimportant.

Any approach to the understanding of problems and how to solve them is engineering. Sometimes it's very structured, sometimes it deals with the physical world - and other times it's incredibly abstract and divorced from the real world. It's all engineering.

streetcat1 70 days ago [-]

There is not such thing is "partial" engineering.

Engineering is the creation and ongoing management of any real world object (building, cars, bridge, computer) which has affect on its environment. I.e. they are placed in the real world to achieve some sort of a function.

The role of the engineer is:

1) Find out what function a machine should provide

2) Create this function by building the artifact on time, and in cost effective manner.

An artist on the other hand:

1) Does not care about the function. I.e. if the thing is useful or not.

2) Does not care about the time/cost etc.

The fact that software is just a description (bunch of text files), does not mean that it does not affect the real world.

As I said, the machine in the real world is the general propose computer. The computer understand this text files, and turn itself into the machine.

Ozzie_osman 70 days ago [-]

I didn't say anything about partial engineering. Just that engineering disciplines are different. If you need to design a column to hold up some load, a set of equations will tell you the dimensions of that column, how much cement, sand, etc to put in the concrete mix, how many steel rebars of what diameter and specification. Designing that column is *very* different than anything I do as a software engineer.

nitrogen 69 days ago [-]

But, if you need to design a data processing system with a certain throughput, a set of equations can tell you how many servers you will need, how much storage, constraints on latency or network bandwidth, etc. So it's not so different after all.

temac 70 days ago [-]

Your layering does not really have practical consequences that would matter to exclude a field for engineering. Plus you are focusing on the technologies, and while this is important this is really not what distinguish an engineer from a non-engineer technician (even an extremely skilled technician), and that part have way more things in common regardless of the field.

2_listerine_pls 71 days ago [-]

Engineering is the application of science. A civil engineer uses laws regarding compression, stress, etc... to build its buildings. If you apply scientifically derived principles when building software then You are a software engineer.

criddell 71 days ago [-]

That depends on what jurisdiction you are in. The title of engineer is protected in lots of places and the description of software engineer is very specific and effectively regulated. That is, a software engineer is licensed.

Ozzie_osman 71 days ago [-]

I'd love to see what those scientifically derived principles are. If I'm building a bridge, there are equations based on physics I can use. Like I mentioned in a separate thread, software is far more removed from those physics.

hyperpallium 71 days ago [-]

GP may be implying from the conditional that there aren't any software engineers.

eitland 70 days ago [-]

While not exact physics we have plenty of real world constraints that we skillfully (or not) work our way around or use for our advantage, here are two :

- Amdalhls law

- Moores law

proc0 70 days ago [-]

"Engineer" Origin: Middle English (denoting a designer and constructor of fortifications and weapons; formerly also as ingineer): in early use from Old French engigneor.

Also shares same root as Engine I assume. Engineers are experts in system building or some kind, the earliest being weapons, then transportation, etc.

Software Engineer is 100% an engineering role IF you are making what could be called a kind of engine. Prime example is working on a game engine's source code, and design (i.e. Unreal, Unity...), or working on a custom operating system. I've seen many people try to downplay how much software has these "engine-like" properties, where you have to pay attention at the "moving parts" (runtime), and those people simply don't see the invisible machine that's doing the work.

WalterBright 70 days ago [-]

Much programming is more art or craft than engineering.

For example, what separates a mechanic from an engineer? They are often working on the same project, and appear to be doing the same thing. In my view, a mechanic follows rules of thumb, eyeballs a design, etc. An engineer applies math and derives design, or checks the design using math. A mechanic will say "this beam looks thick enough to hold the load", and engineer can do a calculation to select the correct thickness for the load.

For software, a programmer can tell you that quicksort is faster. An engineer can tell you why it is faster, and be able to calculate how much faster it is.

anon9001 70 days ago [-]

That's a pretty bad example because any good programmer will have at least a rough idea how to profile code in their environment.

Can you name any software that's "engineered"? I can't. All software is built by craftsman of varying degrees. Nobody sits down and engineers anything up front.

The original point "agile" was that code is an iterative craft, so everybody should get together and work toward incremental improvements and respond to change, rather than try to engineer the whole project upfront. In other words, the way software is "supposed" to be built in most places these days is explicitly the opposite of engineering.

* Individuals and interactions over processes and tools

* Working software over comprehensive documentation

* Customer collaboration over contract negotiation

* Responding to change over following a plan

Doesn't sound like "engineering" to me, and that's the process with the most adoption (even though it's bastardized now, that's still the idea that's being sold).

naasking 70 days ago [-]

> Can you name any software that's "engineered"?

Most compilers. GPS and mapping systems. Basically, software whose core operation founded on well understood data structures and algorithms.

No doubt there is heuristic fiddling around the user interface and that's not engineering, but the core of plenty of software ought to be well engineered.

hyperpallium 69 days ago [-]

I'm interested. Could you elaborate on "most compilers", with a specific compiler, and how it is "engineered"?

Although much care and thought goes into compilers, and they *are* related to discrete mathematical models (though, more descriptive than prescriptive for hand-written recursive descent parsers; unlike the central PDEs in some engineering), there is much tweaking for efficiency and especially for helpful error messages. I would describe that as "crafted" (though "engineering" has some crafting too).

Even something like *diff*, based on longest common subsequence, is full of ad-hoc tweaks to give excellent performance on typical real-world input, much better than theoretical complexity. (I mean the specialization that is in addition to the general optimization in the implemementation of the algorithm.)

I suppose your concept of "core operation founded on well understood data structures and algorithms" is the best we can hope for.

naasking 69 days ago [-]

CompCert is the most extreme example of course given it's fully verified and the strictness required entails a high level of attention to details. Almost every compiler will be based on graph or tree transforms; Haskell and Rust are both engineered around reduction to intermediate, lower-level languages.

I don't think the tweaks you mention necessarily negate the core engineering, unless the system is completely unusable without them or they define more of the behaviour than the core itself.

Edit: for an analogy, consider whether the ergonomics of where buttons are placed on the dashboard of a car suddenly negate the fact that the car is engineered. That seems silly right? Any number of performance or other tweaks can be made to a car and not negate its engineeried quality, until you start compromising the actual operation of the car itself, like swapping out wheels for a tank track, or something like that.

hyperpallium 69 days ago [-]

Thanks, I was thinking of more used languages, such as c, python, java, javascript (they'd have greater *need* of good engineering, wouldn't they?).

I don't know Haskell or Rust well enough to appreciate their compiler engineering (and haven't heard of CompCert). But I might have a look at their source - I have a long-term project about tree transforms ATM and hadn't thought of looking at what compilers do (even though I was aware of Knuth's attribute grammars).

Yeah, nah, I think your "core operation" is a reasonable definition, both as primary function, and as the basic design (despite crafted tweaks atop).

hyperpallium 70 days ago [-]

I agree with your argument, but can you predict performance of an algorithm from big O complexity? (I assume you mean)

Perhaps a few runs of data points could enable curve fitting as it converges.

I feel like the "physics" of software is immature, closer to math than to practice, than for engineering - or something like that.

So I'm also agreeing with the gist of the other replier, that software isn't yet mature enough for precise estimates. We still flounder on what's needed, and stumble around what kind of a thing to build.

Precise specification is the exception.

snorkasaurus 70 days ago [-]

Or in a non-medieval context, an engineer is member of a professional association of engineers whose rights and responsibilities are legislated. And imo, anyone who calls themselves an engineer but isn't a member of a professional association of engineers is misrepresenting themselves.

temac 70 days ago [-]

Highly depends of your region. In France being an "engineer" means having an official Engineering Degree, or simply occupying an engineering position (the degree is not strictly necessary, you can have another one or even none at all).

The common ground of engineers throughout the world is more simply that they practice engineering... (and yes, this has a meaning -- managing requirements and developing in accordance to business values and economics comes to mind -- and yes, this can be applied to SE, and this is even quite "simple" to apply, at least this is not more difficult than for any other field)

I don't really see why some people pretend all the time that SE is different or young and it is not proper engineering. I never read a convincing argument about why it should be so. Of course they are differences but they also are differences between other fields.

proc0 70 days ago [-]

That's like saying no one can be an artist because they're not a member of the official artist club. I'm sure the term engineer outdates any professional association.

snorkasaurus 70 days ago [-]

Actually it's just being respectful. I'm not an engineer, but I work with a lot of engineers who actually have the qualifications to call themselves engineers, so if I call myself one, I'm essentially lying.

SAI_Peregrinus 70 days ago [-]

I am an Engineer, in that my degree is in engineering, and my job is in engineering.

I am not a Professional Engineer, in that I have not taken the Fundamentals of Engineering exam, passed, worked for at least 4 years under a Professional Engineer, and then taken the Principles and Practice of Engineering exam. So I can't put EIT or PE after my name, stamp projects which require it, or operate as a PE in any state.

snorkasaurus 69 days ago [-]

I misspoke then. I would call you an engineer without your being a member of a professional association. I guess in my mind the degree is enough. I'll never have PEng on my business card, but you at least have the potential.

temac 70 days ago [-]

> Software Engineering is not an engineering

For those who refuse to practice engineering, casting that self-fulfilling prophecy over and over instead, yes, what they call "Software Engineering" is not engineering.

They are tons of people who practice Software Engineering as proper engineering for decades, as most people (everybody?) should do.

At least take an other word if you want to designate your practice related to software development that is not engineering, and define it in detail.

perlgeek 71 days ago [-]

I concur.

I'd like draw a parallel to philosophy: There is not one Grand Unified Philosophy Of Everything, but there are many different philosophers you can read, and take some of their views as their own, and still be aware of the rest.

I'm sure Fowler has been in contact with way more projects in enterprise software than me, so I appreciate his perspective. That won't stop me from reading folks from the more functionally inspired crowd, from the DDD corner of software design, etc.

vikingcaffiene 71 days ago [-]

Thanks for this. The initial wave of hate on this post had me worrying that we are collectively forgetting the lessons of the past. This industry continually re-invents the same concepts over and over again and just re-brands them. Once you recognize that, the writings of Fowler & Co become highly applicable to many scenarios.

sidlls 71 days ago [-]

Is this because what they've written truly works or is it because they've simply managed the same thing as the loudest engineer in the office, at larger scale?

vikingcaffiene 71 days ago [-]

Well HN knows a thing or two about the latter ;)

It’s because I write better software in a team setting when I apply the techniques and philosophies of Fowler & Co. The biggest complexity in software is people. Coordinating them, navigating egos, and leveraging their experience levels. It’s a hard problem and the place where I see the most consistent system problems.

IMO Fowler is just providing accessible common sense strategies for dealing with that complexity.

atoav 71 days ago [-]

Interestingly enough this social factor is also the hardest to deal with in other scenarios (expeditions, film crews, multi-party-projects, ...)

geofft 71 days ago [-]

> If you can't even do THAT, then sorry, all your valuable experience is way less useful than someone like Martin Fowler, because I'm not learning anything from you.

There is a small gap in this reasoning: that something is written down and sounds like a good idea doesn't make it a good idea. If you select on "people who have successfully written books about software architecture," you're in no way guaranteed to select for actual good ideas. It's quite possible none of the advice works.

That said, your broader point is taken, and I'll think about how to write things up about actual design lessons I've learned in practice, so still thank you for calling this out.

Scarbutt 71 days ago [-]

There is a small gap in this reasoning: that something is written down and sounds like a good idea doesn't make it a good idea.

That's his point, doesn't matter if is a good or bad idea, but that someone took the time to sit down to think through, write and publish some ideas that will give developers a new perspective on how to do things that they would never thought about, whether those ideas are good/bad, apply or not to the software we are currently creating is up to us to analyze and decided.

bradleyjg 71 days ago [-]

Suppose we weren't software engineers, but instead bridge engineers or physicians. Would you then accept "someone took the time to sit down to think through, write and publish some ideas" that we could either take or leave depending on how persuasive a writer the author happens to be?

I think it is entirely reasonable to ask "what reason to we have to think this is good advice"?

reallydude 71 days ago [-]

Perception, Comprehension, Analysis, Publication repeat.

Somewhere in there, it's doesn't matter if it's "advice" or whatever you want to call it, as long as it's shareable elucidation you can apply your own reasoning...but if you don't bother to follow the pattern, it doesn't matter what you think. Asking someone else to have done these steps for you, is a proxy for a lack of interest, imo. Everyone wants to be lazily told things they either already know or can agree with easily, but being critical in a constructive way is work.

TeMPOraL 70 days ago [-]

I think the point of both 'geofft and 'bradleyjg is that these writings are mostly opinion pieces, and it's justifiable to not trust them based solely on popularity and author's status - none of which involve anyone actually verifying the soundness of all this advice. It's also justifiable to ask for more concrete evidence, since an ethical responsibility of an author is to not waste people's time.

If you were building a bridge, you wouldn't accept that you have to use particular type of beams *just* because a certain well-known bridge architect Fartin Mowler recommends it, and tells stories of how good bridges are when they're using these beams. You probably wouldn't discard the advice either, but you'd ask for more detailed, first-principles explanation, you'd ask to see the plans and the studies of the bridges with these beams, etc.

acje 70 days ago [-]

I think the article is really on to something, but fails to make it clear.

Her is my current belief system;

A) Architecture is strategy. Ted Neward was my first inspiration for this. It makes sense particularly when considering systems with long live cycles. Less obvious for smaller designs that don’t se much change over time.

B) Strategy is most clearly defined by Good Strategy Bad Strategy: The Difference and Why It Matters Book by Richard P. Rumelt Describes the kernel of strategy as

1) diagnose

2) guiding policy

3) coherent action

C) diagnosing is difficult, but both wardley maps and cynefin can help direct us to methods of diagnosing. Unfortunately the immaturity of computer science puts a lot our subsystems in the complex adaptive domain. Partly because we haven’t discovered the best patterns and partly because our tools them selves are a mess of abstractions.

Scarbutt 71 days ago [-]

I remember struggling with my first non-trivial CRUD web app using no web framework, just libs, it had to have a CLI frontend too, his PresentationDomainDataLayering[1] really helped me in refactoring the spaghetti code I had. Probably obvious to many experienced devs but it was enlightening to me.

[1] https://martinfowler.com/blik/PresentationDomainDataLayering...

mobjack 71 days ago [-]

Software development is as much of an art as a science.

Personally I've seen people double down on bad design decisions early in a project that becomes major pain to deal with later on. It is easy to see that if they spent a little extra time on quality in the beginning it would pay off in a few weeks or even sooner.

I also see others spend too much time trying to be perfect in the beginning and nothing ever gets delivered or the over engineered solution creates its own form of tech debt.

Every situation is different so you need to use judgement whether to apply someone's advice.

goostavos 70 days ago [-]

>need to use judgement

I'd put this as the only real Golden Rule of software development. There is so much blind cargo culting around best practices / code / architecture in our industry that, with *some* engineers, human judgement seems to have been replaced with someone else's paint-by-numbers rules. Sometimes, man, it just makes sense to take a step back, look at what you're actually building, and ask what makes sense _for this case_. Sometimes the stupid thing is the best option. Sometimes it's not. Judgement should trump all else.

crimsonalucard 71 days ago [-]

Software development doesn't need to be an art and it doesn't need to be a science either.

By science I mean using data and statistical evidence to drive development, by Art I mean using your gut and your intuition to drive development.

A computer is a deterministic system. It is 100% predictable and follows an exact set of rules. Software development is amenable to proof, logic and theory. You don't need to make decisions driven by data or your gut if the problem space can be described by a logical and axiomatic theory.

The problem is that the current "theory" is incomplete and not all parts of systems are amenable to statistical measurement. Thus a proper system architect needs to unify Theory, Science and Art to form a cohesive framework of system architecture.

Theory is priority because it says definitive things about the system, Science is secondary and operates as a data driven backup for places theory is currently incomplete, and Art is basically the catch all admission that there is no data or theory to back up the problem space and you're just going to have to rely on your intuition.

The problem with Martin Fowler is that all his stuff is 100% Art. It's just made up stuff from intuition and buzz words. Art is largely ad-hoc and does involve a lot of guess work that is often wrong. People identify with Martin Fowlers stuff because they identify with it on the "gut" level. It "feels" right. Your feelings are not a good way to do engineering, you rely only on intuition when you have no choice.

Fowler is also a big Object Oriented Architect. The thing with Object Orientation is that it has no basis in theory or science and is almost 100% art. The pattern is starting to fall out of favor because people are realizing that they tend to build over-complicated systems when following the pattern because they are largely just relying on their gut intuition to build these things using primitives that are validated by the same flawed intuition.

streetcat1 71 days ago [-]

> The thing with Object Orientation is that it has no basis in theory or science and is almost 100% art.

Absolutely incorrect.

The data part of OO is based on set theory as formalised in the database design research and type systems.

The dynamic part of OO is based on finite state machines.

The data abstraction and information hiding part is based on the work of barbara liskov (for which she won the Turing award) - https://en.wikipedia.org/wiki/Barbara_Liskov

crimsonalucard 70 days ago [-]

>Absolutely incorrect.

I said OO didn't have basis in theory or science. By basis I mean it wasn't derived from theory. Rather it was created as more of an "artistic" endeavor and later it was fit into a framework of an existing theory. You should know almost ANY system can be broken down to fit into an existing theory from a projectile object, to a flying plane, to the weather, to an object oriented programming language. Thus following that line of thought ANYTHING can be a part of a theory but I use the term "basis" to emphasize that something can be derived out of theory.

>The data part of OO is based on set theory as formalised in the database design research and type systems.

I highly disagree. Type systems are not exclusive to OO, additionally the OO type system allows for mutation which is basically not at all part of set theory at all. Also type systems and database systems didn't formalize set theory. Set theory comes from mathematics.

>The dynamic part of OO is based on finite state machines.

This I can agree. A state machine modifies a change of state which is closer to the definition of an object in OOP.

>The data abstraction and information hiding part is based on the work of barbara liskov (for which she won the Turing award) - https://en.wikipedia.org/wiki/Barbara_Liskov

This has more to do with the type system then it does with OOP. Also note if you read the paper. Her paper has no data, statistics, formal theory, rigor, theorems, proofs or axioms. So although she won the turing award, although this is a research paper... it is largely "artistic" in the sense I defined it. See the paper here:

http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.136...

Note that her paper is important, I am not discounting it. But it is largely more in the realm of design then it is in the realm of theory and science.

streetcat1 70 days ago [-]

She developed the axiomatic theory in a later book:

https://mitpress.mit.edu/books/abstraction-and-specification...

Were she propose an axiomatic theory for OO interface design.

Definitely not artistic.

The only thing artistic in OO (as someone who practice it in the last 30 years), is finding the initial set of objects. But you will have this issue with any technique.

crimsonalucard 70 days ago [-]

>The only thing artistic in OO (as someone who practice it in the last 30 years), is finding the initial set of objects. But you will have this issue with any technique.

OO is highly artistic. It's not just the initial set of objects you decide on, but you have to decide how each and every single object composes and which objects compose. There's multitudes of techniques for how to compose objects and when to compose: Inheritance, dependency injection, Object composition, etc, etc. Within each of these techniques you still have to write custom code, what is the parent, what is the child? What property overrides what? How does the parent use the injected object? etc etc... You are literally designing the system rather than operating within a theoretical framework. It is largely art.

If you operated within the framework of an existing theory you would be able to much better predict the outcome of a system at scale, but because in OO you are creating the system from scratch for almost every application it is largely unpredictable.

In mathematics all theorems flow from your choice of axioms. There is little room for deviation.

Interface is not exclusive to OO. It's a seperate concept. I don't have access to that book so I can't say what she talks about in the book specifically but an interface is not exclusive to OO. The concept exists in mathematics as well which deals more with functions and types rather than objects.

Additionally OO to my knowledge came later after her paper on abstraction. Alan Kay developed OO. Then OO as we use it today came when JAVA and C++ grafted parts of the concept into the framework of procedural languages. This is not deriving a language from theory, it's more of an organic evolution of concepts. This isn't bad, but it's not axiomatic, scientific or theoretical. It is art through and through.

As I said, any system artificially created or not can be fit into the framework of some theory. There is a huge difference though between systems that are designed artistically vs systems that are derived from theoretical knowledge.

perlgeek 71 days ago [-]

> A computer is a deterministic system. It is 100% predictable and follows an exact set of rules. Software development is amenable to proof, logic and theory. You don't need to make decisions driven by data or your gut if the problem space can be described by a logical and axiomatic theory.

That's true, only if you can assume perfect knowledge about the problems/requirements, the data, the environment, and how (if at all) those will change.

In practice, none of these tend to be true, and that is where a lot of gut feeling will come in.

An experienced practitioner will just *know* that certain modules / feature sets will grow and grow and grow over time, and will take special care to build a particular robust approach for that area. Or that certain kinds of data are often of low quality.

crimsonalucard 71 days ago [-]

Yes this is where the "Art" that I described also comes into play. We have no theory that can predict the output of a human brain therefore we can't theoretically derive what requirements a customer will want or change over a span of time.

However...

There is something sort of new from the academic side that could potentially be a theory that can deal with problem you describe.

At a high level the theory looks almost like a formal language for "design" and "abstraction." It's called category theory. It's new-ish so it doesn't yet solve the problem you describe but I see the outlines of it within the framework of the theory.

lidHanteyk 70 days ago [-]

Category theory is nearly a century old; what are you talking about? People write code today to solve these problems, and it's category-theoretic code: https://www.categoricaldata.net/fql.html

crimsonalucard 70 days ago [-]

Yes yes, you are right. I mean new-ish in it's application to things outside of math. And also very new to programming.

In terms of just pure math though, category theory is very new when compared to most of the disciplines around today.

Also you are completely wrong about it being a century old. It was invented in the 1940s.

I'm assuming you just googled it to find counter points. That being said you missed something even OLDER then CQL and the most popular and definitive use of CT in programming today:

Haskell.

Anyway my main point is that Category Theory can answer some questions about 'deriving' an architecture for a problem rather than having a person 'design' one. The way category theory is used and haskell and CQL is as a framework. In haskell and CQL it's more about designing the system within the constraints of category theory then it is about deriving a system. The derivation of a system from a formal theory was ultimately what I was trying to get at.

You can sort of see an outline for the possibility of this happening when you study CT. I'm being really speculative here in case I failed to convey that. Such a system doesn't actually exist (yet). That's all.

lidHanteyk 70 days ago [-]

Haskell is not really category-theoretic. The category Hask doesn't quite line up with real-world Haskell. The disconnection is well-understood and well-documented [0][1].

Category theory can directly give us systems from theories. See the papers behind FQL/CQL for more details, especially [2] and [3].

[0] https://wiki.haskell.org/Hask

[1] http://www.cse.chalmers.se/~nad/publications/danielsson-et-a...

[2] https://arxiv.org/abs/1102.1889v2

[3] https://arxiv.org/abs/1706.00526

crimsonalucard 70 days ago [-]

Well you're not technically wrong about the Haskell thing. I'll look into FQL and CQL. If such a thing exists, what in your opinion is stopping an architect from using it?

Is it just that it's not popular or are there flaws?

lidHanteyk 69 days ago [-]

If you use it now, and it ends up being popular like Java or Python or Haskell are popular today, then you can expect a two-decade delay before you're vindicated.

perlgeek 70 days ago [-]

What's the relation between category theory and changing requirements? You lost me there...

crimsonalucard 70 days ago [-]

An example:

Since category theory looks almost like a formal theory of "design" and "abstraction" is looks like if one day someone could come up with an algorithm or function to formally derive a solution given a set of requirements. To anticipate changing requirements you simply need to adjust the initial set of requirements to be more broader and then this hypothetical function would output a system that can handle potential changes that you anticipated.

This is all hand-wavy and not concrete. Many people who study CT see it as currently the closest thing to a formal theory of "design" and "abstraction." If this is true then techniques or algorithms for deriving architectures of systems can become turn key.

Of course let me emphasize, I'm just speculating here. Not talking about anything that actually exists. Just saying that if you study CT you can kind of see this kind of thing existing sometime in the future.

krapht 70 days ago [-]

Proving diagrams commute with abstract nonsense is not going to help the average programmer. At most it's a formalism that will help people who design programmer's tools reason about their system, much like relational algebra for databases or Hoare logic for imperative programming, or UML.

crimsonalucard 70 days ago [-]

Relational algebra is more or less isomorphic to SQL if you're doing SQL you're doing relational algebra.

Hoar logic can prove your programs 100% correct. This is useful for anyone who wants to write a program that's bug free.

UML is more of a visualization tool then a theoretical tool.

>Proving diagrams commute with abstract nonsense is not going to help the average programmer.

What in the world is a proving diagram, and what is abstract nonsense? Please be more clear with your explanations. I am not talking about diagrams and trying to say they commute with anything.

alanfranz 69 days ago [-]

> People asking for proof: what sort of evidence would convince you? I take these blog posts, apply it to my experience, and take what I think makes sense.

I love opinions. But if you provide a plot like the one for software quality, you should provide backing data. OR you should clearly state that such plot is totally made up. Nothing else.

I have a blog where I write about my opinions. But they're CLEARLY opinions, UNLESS I have some backing data.

blub 70 days ago [-]

If you want to see how a properly referenced text looks like, pick any of the following:

* Code Complete - McConnell

* The Economics of Software Quality - Capers Jones

* Facts and Fallacies of Software Engineering - Robert Glass

* Making Software - Oram & Wilson

... and there are many others. Not all references have survived the test of time, L. Bossavit describes in his book " Leprechauns of Software Engineering" how several software development "facts" are in fact myths.

The approach you suggest suffers from a chicken & egg problem because you first have to be experienced enough to be able to evaluate such reference-free content, as the blog written by Martin Fowler. It's precisely the inexperienced that are likely to internalize such advice and perpetuate personality cults.

Your objections are invalid; one does not need to be better at a task than an individual in order to criticize them. Asking for references and proof is perfectly legitimate, at least if we want to approach our discipline more as engineering and less as "Strunk & White" as someone comically suggested in another comment.,

Barrin92 70 days ago [-]

>Your objections are invalid; one does not need to be better at a task than an individual in order to criticize them. Asking for references and proof is perfectly legitimate, at least if we want to approach our discipline more as engineering and less as "Strunk & White" as someone comically suggested in another comment

Software development, as often as comparisons are made to engineering or science is neither. It's a craft. The tools differ from person to person, each piece of software is different, a lot of practises are learned and tacit and transmitted from one pracitioner to the next.

Writing code is more like being a blacksmith than being a scientist, as much as people yearn for formalistic answers and peer reviewed studies so that they can have the appearance of objectivity, a lot of software development is fundamentally subjective.

mlthoughts2018 70 days ago [-]

The rampant semantic games about art vs science vs engineering vs craft littering this thread are nauseating. It’s all of those things in different contexts and situations, and regardless, seeking evidence backed advice is the right thing to do no matter what context you’re in.

My electrician father doesn’t base his craft on thought leading blogs. He has actual textbooks on electrical equipment, and guides with examples on using tools and solving certain common problems, full of reference material to other books.

I happen to think software is something like 70% craft, 25% science and 5% art, and to me this suggests *not* putting much stock in armchair blog opinions (and yes that includes Fowler’s) and instead seek scientific or data-backed analyses, *even for craftsmanship* such as any of the references in the comment above yours.

blub 70 days ago [-]

Software development does have a certain Darwinian character - whatever survives and thrives can't be wrong.

Embracing this seems misguided, because as software becomes more complex, it's no longer enough to have software blacksmiths banging away with their software hammers and unit testing the end result into a releasable shape.

Teams working on high-availability, high-reliability software know this and take a rigorous approach. Too many take the BDD approach (blog-driven development) and when the thing inevitably falls apart, the customers pick up the broken pieces.

atoav 70 days ago [-]

> Software development does have a certain Darwinian character - whatever survives and thrives can't be wrong.

I am not sure of that. A lot of what we see today is just adding an abstraction on top of other abstractions. People use five frameworks for a static webpage that could have been easily made in half a day using editor, css and html without ever writing a single line of javascript.

In a evolutionary sense it *works* for them, but the reason isn't that html and css wouldn't also work, the reason is that they believe they *have* to use it.

So I'd be careful with the darwinian comparison here, this is more a mimetic or meme-like evolution of ideas and once the layer below is abstracted away it is quite easy to just forget about it alltogether.

rhizome 70 days ago [-]

A lot of what we see today is just adding an abstraction on top of other abstractions. People use five frameworks for a static webpage that could have been easily made in half a day using editor, css and html without ever writing a single line of javascript.

Not to speak for them, but I think it's more that despite decades of ridicule, there's still a lot of COBOL out there humming along. Some of it might even be deciding whether or not your health insurance is going to cover that thing that you went to the doctor for the other day. See also: Visual Basic, ColdFusion and so on.

onemoresoop 70 days ago [-]

Not sure I agree. One example. Theres a lot of mainframe code still in production, it survived and it’s still doing its job to this day. The reason it survived is because replacing it is very very costly.

inopinatus 70 days ago [-]

> Teams working on high-availability, high-reliability software know this and take a rigorous approach

Approaches that are, nevertheless, still more akin to philosophy than engineering.

blub 70 days ago [-]

Please detail your understanding of what philosophy is and how it resembles what such teams are doing. My gut feeling is that they're quite different, but I could be missing something...

inopinatus 69 days ago [-]

I don’t reward sealioning.

Chris2048 69 days ago [-]

> Sealioning is a type of trolling or harassment which consists of pursuing people with persistent requests for evidence or repeated questions

Asking what you mean by what you wrote in a public discussion forum isn't seasoning. You aren't being asked for evidence for you opinion, but what that opinion *is*, and the context of a discussion forum implies you mean to discuss the topic. Also, I can't see the "persistence" or bad-faith that would make it trolling.

inopinatus 68 days ago [-]

I looked at the correspondent’s other posts in the same thread. They qualified.

blub 69 days ago [-]

All right, but if we take the standard definition of philosophy then your initial claim makes no sense.

The study of the fundamental nature of knowledge, reality and existence is unrelated to writing software.

inopinatus 68 days ago [-]

When I build applications the questions of “What is truth?” “What is the essence of a thing?” “What is useful?” are top-of-mind throughout, because there’s no rigour without context and purpose.

Software development is applied philosophy. Wittgenstein would’ve been good at it.

Chris2048 69 days ago [-]

Software development is indeed not "engineering or science";

But that doesn't mean it has to be "craft" or other highly subjective fields, nor that we can't aim for similar standards as other engineering disciplines.

> Writing code is more like being a blacksmith than being a scientist ... a lot of software development is fundamentally subjective.

But writing code is more like engineering than science, so why not compare those? There is subjectivity to engineering design too, but plenty objectivity in the constraints - what make something "fundamentally" subjective versus some other kind of subjective?

Is it possible we don't need to compare programming to any other profession? We can instead identify there is subjectivity in software development, and that some (but maybe not all) it it could be more objective for a benefit.

hutzlibu 70 days ago [-]

"Your objections are invalid; one does not need to be better at a task than an individual in order to criticize them"

He did not say that. He said, that he wants concrete criticism, what ideas are wrong and if possible, why ...

inopinatus 70 days ago [-]

Referencing and evidence has never been a benchmark requirement for works of philosophy.

mcnichol 70 days ago [-]

There are many arguments that sound logical on their face but completely miss the mark.

Laffer curve is one that comes to mind immediately. The saying goes: "It can be explained to congress in 6 minutes and used for 6 months."

It can be argued (with data) that it failed.

Martin Fowler is no doubt a smart and greatly respected person in this industry.

Saying something qualitatively doesn't mean there isn't quantifiable data behind it...but with his vantage point I believe we should press for a higher source of truth.

I assume he can understand how important that higher source of truth is in having these conversations regarding what moves the architectural needle.

Else we stay in this echo chamber.

2_listerine_pls 71 days ago [flagged] [-]

ok, then. I will ask for no proof and believe in anything He writes.

dang 70 days ago [-]

Please don't post unsubstantive comments here.

cliffcrosland 71 days ago [-]

Reading an enumeration of architecture options can be helpful, but I've probably found more value in works like Ousterhout's "A Philosophy of Software Design" that cover fundamental design principles. Such principles seem to be applicable no matter the project.

I appreciate that Ousterhout tries to support these principles with some data from his own experience (building Tcl, RamCloud where Raft consensus was discovered, etc) and from code reviews of student projects in his design course, but he admits that there is a lack of data about things like design.

Sample principles from Ousterhout:

* When developing an interface, favor exposing a small number of functions each with deep functionality over exposing a large number of functions each with shallow functionality. For example, favor Unix file I/O interface (open, close, read, write, unlink) over Java file I/O interface (File, FileReader, BufferedReader, FileWriter, BufferedWriter, FileNotFoundException, ... etc.)

* When choosing between keeping your code specific and keeping your code general, favor "somewhat" general. For example, instead of exposing "deleteNextCharacter(text, pos), deleteNextWord(text, pos), deleteNextSentence(text, pos)", just expose "delete(text, pos, length)" and let the caller use it how they will.

blub 70 days ago [-]

Yes, "A Philosophy of Software Design" is brilliant, precisely because the author supports their statements through the many experiments they performed with students and

The fact that they're the author of a well-known piece of software helps, although we should not use this as the main criteria to evaluate a text.

Offering at least evidence based on experimentation is essential if we want to move past personality cults and having entire teams work based on what someone wrote on a blog somewhere.

sriku 70 days ago [-]

I'm wondering whether another factor that might be at play is that as consultants they may not be at liberty to discuss that kind of detail. Barring that, it would be interesting to have Fowler sit in Ousterhout's class, take on an advanced problem and see what they come up with.

vikingcaffiene 71 days ago [-]

I gotta say I am surprised and a little disheartened by all the negativity around this post and Fowler in general. Fowler's writing has been a huge inspiration to me in my career. I'm about to gift a copy of his book "Refactoring" to one of the junior engineers I work with for instance.

I think the idea of an architecture as a social construct is very fitting. Software is hard when multiple people are working on it with differing backgrounds and experience. Having a rough outline (emphasis on rough here) of "what lives where" and "what is allowed to talk to what" is absolutely critical IMO. It's how your team can effectively reason about a system and add new features to it. It should also be a living thing in that it's something that is continually refined, discussed, and changed as the needs of the business change. Done right, its an organic process that really helps deliver more robust software that everyone involved has a high level understanding of.

Contrast that with an Architecture (capital A) done in Enterprise which delivers a static artifact that does not allow for any creativity and micromanages every aspect of it. It's split off into chunks and no one without an expert level of knowledge can easily understand it. That kind of design will almost certainly fall over the first time there needs to be a change to it. It's important to point out the difference because that is NOT what Fowler is recommending in his writings.

wpietri 71 days ago [-]

Totally. In a side room at Java One in 1999 or 2000 I saw Fowler talking about Refactoring, and he happened to bring Kent Beck up on stage to talk a bit. If I hadn't been exposed to their ideas, I surely would have left software by now.

Over and over, my experience was building something that slowly got less tractable over time, where eventually we'd just want to do a big rewrite. It sucked! I had tried the big-design-up-front approach, where I thought real hard for a long time before building. That might push out the rewrite date some, but definitely not forever. It was dispiriting, a slow, sticky death. That combined with crunches and burnout was slowly making me hate something I had loved doing.

But their main point was that code bases didn't have to end in an entropic snarl. That with decent tests and a little elbow grease, code could get better organized every time we touched it. That instead of death by papercut, we could make things a little better every day. Better for the users, better for the business, better for the developers. They weren't promoting silver architecture bullets. They were saying that we should trust and empower teams to figure it out. That if we gave them room and tools to figure it out, everybody would end up happier.

It made a huge difference in my life, so it's weird to see people here aggressively putting words in Fowler's mouth, words that seem entirely at odds with my experience of him and his work.

vikingcaffiene 70 days ago [-]

Thanks for taking the time to write this. I really hope that the latest generation of developers don’t see all the negative HN hive mind comments and discount writings like this.

jammygit 70 days ago [-]

I find Fowler writes/speaks more clearly about topics than anybody else. A few of these topics like refactoring or RESTful design have been very helpful for me.

EdwardDiego 71 days ago [-]

> All I see is yet another reasonably sounding yet unsupported piece of folklore.

Another artefact of the fact that software developers are still craftspeople - as in "someone skilled at making things by hand".

Software development output is something that no-one has yet successfully quantified. There's no easy metric to assess a non-productive software developer from a productive one. We can all tell the difference, but good luck measuring it. In my country, which (thankfully) lacks America's "at-will" employment contracts, it does mean that it's quite hard to fire an underperforming dev, because how the hell do you prove they're underperforming? What metric do you use?

Bug rate is a bad one, because very smart people working in a new problem domain may register an significant uptick in bugs - doesn't mean that they're incompetent, it's just the result of doing new stuff in an unfamiliar domain in a highly complex environment.

Lines of code? The less said about that metric and how gameable it is, the better.

Us still being craftspeople is also why we can command such high wages relative to the rest of the economy - what we do is not yet automated, so we can still draw a premium.

noobiemcfoob 71 days ago [-]

> it does mean that it's quite hard to fire an underperforming dev, because how the hell do you prove they're underperforming? What metric do you use?

I think your criticism of 'at-will' employment is unfounded in this scenario. Having lower guardrails on employment means it's not as pressing to figure out performance metrics. If the company narrative is that an employee is helpful to the mission, then there isn't much cause to fret for a more perfect metric. And at-will means you can test more employees before committing to a few for longer than a couple years.

EdwardDiego 71 days ago [-]

I didn't express any direct or detailed criticism of at-will in my comment, just my gladness that we don't have it.

Why, despite the difficulties it causes me when dealing with poor performing devs, am I happy we don't have at-will?

Because at-will dramatically distorts the employer - employee social contract in favour of the employer. And employers already have a disparately large amount of power over their employees as it is. So there you go, that's my criticism of at-will employment. It puts the well being of the employer ahead of the well being of the employee.

You can have a trial period without "at-will".

noobiemcfoob 71 days ago [-]

Gladness you don't have it *is* criticism you're trying to hide behind semantics.

You and I seem to have different ideas of an employment social contract (or non-existence of one). Unless you're interested to dig into that difference, I don't think we have anywhere else to go with this thread.

CraigJPerry 70 days ago [-]

Gladness: "I'm glad i don't have a million dollars to manage in my bank account, just getting the best from my small savings is stressing me out!"

Thankfully: "My bank account (thankfully) lacks a Bill Gates' sized stash of greenbacks, it does mean that it's impossible to buy a mansion though"

I'm not criticising the concept of a million dollars in either of these statements. Thankfully, as used in the original context means something quite different to the meaning you've derived.

I might still actually have a bone fide criticism of a million dollars, but that's neither here nor there in the context of either statement.

I think wrapping it up's a fair shout though so no hard feelings and all the best.

wolco 71 days ago [-]

At will doesn't mean you can test out more employees. It means you don't have to pay anyone any pay when you let them go.

It has side-effects. A worker will bolt at the first better opportunity or when your company looks like they are in trouble because they have to. It also raises wages as people changes jobs more often.

CraigJPerry 71 days ago [-]

> I think your criticism of 'at-will' employment

There was no criticism of at-will employment in the parent post.

That's a hair trigger you've got yourself there fella.

noobiemcfoob 71 days ago [-]

> In my country, which (thankfully) lacks America's "at-will" employment contract

I'm sorry, is that not obviously criticism? My parser must be busted again.

geekone 70 days ago [-]

Looks like it is criticism of America's "at-will" labor laws to me too.

CraigJPerry 71 days ago [-]

No, it's not. Which is why I made my post.

noobiemcfoob 71 days ago [-]

Ah, noted. Thanks for the education.

charles_f 71 days ago [-]

I don't think my concern with the issue at end is much more related to automation and wages than it is related to the "engineering" side of things.

Most of what the "literature" recommends is based on individual lessons learned and ideas, that are indeed backed less by data than they are by anecdote.

Having that is better than nothing. But if you compare software "engineering" when talking about software architecture, to construction, electrical or such, it's puzzling indeed to see how much more craft and less actual science is involved.

There are metrics that seem to apply though. Lines of codes and bugs aren't good indeed. Cyclomatic complexity is more related to algorithmic than architecture (which if going that direction is very much a science).

Measuring architecture effectiveness is measuring how much the architecture achieves its goals. These are contextual, most commonly in business they will be related to things like quality (as in, the software does what it is supposed to), total cost, time of execution. Some goals you can derive from that are adaptability to change, ability to modify and extend, which you might derive in turn into qualities such as decouple-ness, non redundancy of code, readability, etc. Some metrics for these exist, such as the instability (fan-in / fan-out ratio) or the abstractness measure of packages.

I don't know of architecture books that present patterns or practices with this sort of justification though. There's a chapter on that in Clean Architecture, but most of the rest is only strongly worded opinion.

proc0 70 days ago [-]

Of course it can be quantified otherwise hiring based on skill level for a software role would be impossible. You can look a examples of code and know someone's level and then deduce their performance based on output and quality of code. The problem is NON-technical people are the ones who want to quantify your performance! Which has been my problem for years in the software industry. If you haven't written code in years, you will be lost as to whether I'm writing good software or not, yet these are the people who rise to the top to manage the technical people and then do a horrible job because they have no idea what to optimize for.

jammygit 70 days ago [-]

The standard practice is to quiz based on algorithms and data structures. How is that quantifying anybody’s performance? I knew a smart young algorithms guy once who could solve anything, but he still liked storing all his data in the global scope until he was told to stop it.

proc0 69 days ago [-]

Yeah now that I think more about it, those quick tests say nothing about a candidate. It's definitely harder than it seems.

jeswin 71 days ago [-]

I looked at this thread a while back and closed the window, hoping that those early comments don't influence the younger practitioners who are on HN.

As someone nearing 40 (mainly doing consulting work around boring business apps for a long time), my view is that Martin Folwer's biggest contribution is indeed the documentation, and defining the vocabulary to enable discussion around frequently encountered (but perhaps mundane) problems in Enterprise Application development. He isn't trying to break new ground, but I'm glad his writings exist. To pick a very random example, here's CQRS defined in an approachable manner[1]. It's a great link to send to a team of 50, out of which quite a few will be fresh out of college. And I am pretty sure Martin Folwer has seen plenty of real world code while he was working at Thoughtworks. The ideas in his writings aren't conceived in some vaccuum.

Some of his early writing isn't perhaps as relevant today. For instance, he talks about the flaws of Anemic Domain Model here [2]. Back in the day, many of us tried to enrich the domain model with particular emphasis on reusability. But those things are less relevant in today's heterogeous environments where logic lives across in multiple services (many outside your team's control). So just reading through his site will surface a lot of outdated ideas; but that's helped a lot of companies along the way when we didn't have to aim for web-scale tps. Actually, it might still work for 99% of apps even today.

HN tilts towards the experiences of Startups and young founders, but one good lesson to learn early is that we should be more open minded.

1: https://martinfowler.com/bliki/CQRS.html 2: https://martinfowler.com/bliki/AnemicDomainModel.html

extra_rice 71 days ago [-]

I don't understand how anemic domain models are irrelevant now especially in a time where a lot of software practitioners seem to observe domain driven design. In the age of microservices, it is very important to determine clear context boundaries. "Logic lives across multiple services" sounds more like a design smell to me. In this case, changing even very simple business requirements could mean changes in multiple parts of the system, which could mean multiple redeployments of many different services. Also implies more complex integration (even e2e) testing.

jbmsf 70 days ago [-]

I'm definitely guilty of using and promoting anemic models. I find that teams have a hard time deciding whether logic is service or domain or whatever and gravitate towards putting everything in one place *if* domain objects are allowed to be at all smart. On the other hand, people do well with the rule that logic belongs in procedural layers with clear names and some sort of maximum size/complexity.

I also find that these discussions matter less in microservices because the size of each service is so small that you usually don't need more than two layers: one for business logic and one for persistence.

I very much agree that microservices create their own challenges at service interfaces, but since these boundaries are usually expressed as http operations instead of object oriented function calls, I look for different solutions, especially various forms of system-wide introspection and testing.

Spearchucker 70 days ago [-]

First, anemic models work well in service contexts. You can't send a business rule from a web page to a web service as part of a data entity. And so such models have a valid place in the world.

Micro services are still services. As such we always validate data crossing a trust boundary - whether monolith or micro service. The rules to do that validation cannot travel with the data.

So guilt because anemic strikes me as... odd. Modern connected apps aren't written as single-tier Smalltalk apps.

lonelappde 70 days ago [-]

You wouldn't send a business rule from a web page, you'd put it in the domain model on the server.

The web page has a UI model, not a domain model.

Spearchucker 70 days ago [-]

That's the point. And yet Fowler objects because it violates encapsulation and information hiding, because it requires a service layer and makes a model less expressive.

extra_rice 68 days ago [-]

You can easily serialise a well fleshed out (i.e. non-anemic) object. Does the view layer in this particular example need to know the encapsulated business rule? In my experience, not usually. Otherwise, there's most likely a problem with responsibilities. And when that data comes back from the UI, it's also relatively easy to reconstitute it back to well fleshed out objects.

Spearchucker 67 days ago [-]

This I know. The point isn't how one might do it, or whether it can be done¹. The point is that Fowler has issues with anemic objects, even though there are patterns in distrubuted computing best solved by anemic objects.

¹ Note that reconstitution does not do away with anemic objects - reconstitution does not implement logic in a purely object-oriented way, i.e. the logic does not travel, but the data does.

Ozzie_osman 71 days ago [-]

Your point on vocab is so spot on. Simply identifying something as a pattern and naming it is tremendously valuable. Doing it in a way that's packageable and shareable with others is not easy at all.

camgunz 70 days ago [-]

Agree, but people tend to stop at Fowler and live it as gospel. There is a profound richness in the different ways to think and talk about computers, engineering, and information science, far more wonderous than any one person can catalog. Read SICP, Knuth, Code Complete, Codd. Hell, just YouTube around. If your vocabulary is prescribed by literally one guy in a fedora, you gotta get out there. Might I suggest Joe Armstrong [1] :)

[1] https://youtu.be/lKXe3HUG2l4

charlieflowers 70 days ago [-]

> Agree, but people tend to stop at Fowler and live it as gospel.

I know people do that, and it sucks. But I never got that from Fowler himself.

There are other authors (not naming names) who are pretty dogmatic often. But Fowler seems to genuinely try to capture patterns and techniques in an open-minded way.

It's the abuse of his material that is the problem.

camgunz 70 days ago [-]

Yeah he's no Zed Shaw. But actually I think Fowler is... well he skips a lot of steps and makes a lot of assertions. Here's an excerpt from the intro to *Refactoring*:

> The performance of software usually depends on just a few parts of the code, and changes anywhere else don't make an appreciable difference.

> But "mostly" isn't "alwaysly." Sometimes a refactoring will have a significant performance implication. Even then, I usually go ahead and do it, because it's much easier to tune the performance of well-factored code. If I introduce a significant performance issue during refactoring, I spend time on performance tuning afterwards. It may be that this leads to reversing some of the refactoring I did earlier--but most of the time, due to the refactoring, I can apply a more effective performance-tuning enhancement instead. I end up with code that's both clearer and faster.

> So my overall advice on performance with refactoring is: Most of the time you should ignore it. If your refactoring introduces performance slow-downs, finish refactoring first and do performance tuning afterwards.

One way to read this is basically what he writes: clear code is more important than efficient code, and clear code can help you make code more efficient if that's important.

Another way--indeed the way I read it--is "performance isn't the most important thing, and at least some of the time following my advice, performance will suffer, even if you undo a lot of the work I'm recommending you do." And I can respect that, but what he writes tries to have it both ways. At least when DHH makes this argument, he basically just says "yeah, buy more machines, they're cheap, engineers aren't." This "incidental" absence of drawbacks is just one of the things that makes all of Fowler's writing feel like marketing.

Another is all the hand waving assertions. There's a lot of it even in those three short paragraphs:

- How did he measure "much easier to tune the performance of well-factored code"? Honestly how would you even measure that?

- How did he measure "more effective performance-tuning enhancement", and also what does "effective" mean?

- What does "well-factored" mean?

- What does "clearer" mean when he says "I end up with code that's both clearer and faster"

Further, he reinvents and renames things. His (short) article about "Application Boundary" [1] is actually Conway's Law [2], which was nearly 40 years old at writing. And OK, no one knows everything. But that piece is written *very* prescriptively; here's an excerpt:

> We can draw application boundaries in hundred arbitrarily different ways. But it's our nature to group things together and organize groups of people around these groups. There's little science in how this works, and in many ways these boundaries are drawn primarily by human inter-relationships and politics rather than technical and functional considerations. To think about this more clearly I think we have to recognize this uncomfortable fact.

How can you say "there's little science in how this works" immediately after "it's in our nature to..."? And for what it's worth, there's plenty of work around this; look at any discussion of permission/capability systems, consensus systems, sandboxes, etc. There is literally science about it.

I have no doubt that Fowler is earnest and well-meaning, I appreciate his contributions to (what I reluctantly accept is) my field, and I'm happy that he's frequently many people's gateway to thinking more critically about software. But he's far from the final word on it, and too often he writes like he is.

And, at the risk of making an over-long post even longer, there are two problems with that. I've already been explicit about the first: people take it as gospel and don't look further. That's not great for our field.

But even worse, it ends up changing our culture from one of wonder, exploration, and experimentation to one of prescription and certitude. I have no idea what the best way to factor a program is; it's very hard for me (or anyone) to say why one expression of information or a computation is "better" than another, and that is an endless source of delight for me. God help me the day I figure it all out.

[1]: https://martinfowler.com/bliki/ApplicationBoundary.html

[2]: https://en.wikipedia.org/wiki/Conway%27s_law

goto11 70 days ago [-]

Are you applying the principle of charity when reading Fowler? The quote about performance seem totally sensible and basically akin to Knuths "We should forget about small efficiencies, say about 97% of the time...". Focus on maintainability and only sacrifice clarity for performance when you *know* there is a performance problem. What part of this do you honestly disagree with?

You can pick any writing apart by being deliberately obtuse. It doesn't help the conversation. E.g rhetorically asking the meaning of "clear" and "well-factored" code, when basically all of his writing are about exactly that.

I'm sure some people are taking him too much as gospel and don't look further - since this happen to any sufficiently popular writer. Just look at all the people on HN which read PG's essays as gospel and think all problems of architecture and maintainability can be solved by using Lisp - you can't really blame PG for that either.

Yes you will have to apply you own experience and critical thinking and understanding of the problem domain. This is true for *everything* you read or hear.

camgunz 69 days ago [-]

> Are you applying the principle of charity...?

Well, when I read that passage, I tried to figure out what he might mean. And mostly I distilled it down to what I wrote: "clarity is more important than performance, and will usually lead to better performance". So even though he's lacking a lot of specifics, sure I think I get his meaning.

> "That's basically exactly what Knuth said"

No Knuth said "ignore small inefficiencies". Fowler said "clarity via refactoring is more important than performance, and will usually lead to better performance". Those are very different.

> Focus on maintainability and only sacrifice clarity for performance when you know there is a performance problem. What part of this do you honestly disagree with?

A good and well known example are entity systems in game engines. If you build such a system using OO principles--polymorphism, encapsulation, messaging--you will have an entity system that doesn't support very many entities on common hardware. It will also have very complex and hard to predict behavior as entities are created and destroyed, and as they all react to different messages, even if you break down and do it all synchronously.

Unless you have a lot of certainty that any game built with that engine will have a (far) below average entity need, you'll end up building the wrong thing, even following best practices, and no amount of refactoring will help you; it's a fundamental design limitation. In fact, you have to design a system that breaks these rules in order to get good performance, and I'm not talking about wonky code here and there; I'm talking about building something that (for example) lets you avoid vtables entirely.

So that's an example of why I disagree with Fowler on refactoring and performance. I think he's basically as wrong as you can be about it. It's a complex subject, and no, it doesn't just fall out of "clear" code, whatever that is.

> You can pick any writing apart by being deliberately obtuse. It doesn't help the conversation. E.g rhetorically asking the meaning of "clear" and "well-factored" code, when basically all of his writing are about exactly that.

I can understand how I probably came across as overly pedantic, but I don't know any other way to pin down all the vagueness. A lot of people find Haskell very clear and idiomatic C unreadable, or a lot of people can read fucked up SQL but can't make heads or tails out of ES6. If I'm gonna burn engineering time on making something "clearer", I'd like some idea of what that is.

And my strong feeling is that it's very situational, contextual, and subjective. What's clear in pixman is confounding in Django; what's elegant in Java is weird as hell in Python. I'd love to read a book that explored these differences and gave me more insight into how to express myself more clearly across them. Refactoring is just not that book at all, and in fact it encourages lots of engineers to apply its concepts everywhere, which of course makes things less idiomatic, less context-aware, and thus, ironically, less clear.

goto11 69 days ago [-]

> clarity is more important than performance,

But he does *not* say that! You are deliberately misrepresenting the meaning of the quote. He says he focus on clarity *first* and only optimize for performance if absolutely necessary. You can disagree with that, but it is a totally different thing.

If you have legitimate disagreements which his viewpoints you should be able to state your argument without needing to misrepresent what he says. Then we can have a conversation and maybe learn something.

camgunz 69 days ago [-]

I mean I quoted the entire 3 paragraphs up there, so of course I'm paraphrasing.

Fowler advocates strongly for OOP. Presumably he'd find an OO entity system clearer (if you're gonna quibble with this, imagine how much more productive this conversation would be if we had an understanding of what "clearer" meant, then recoil in horror as you realize thousands of developers are having this exact same unproductive discussion, all because one guy said he had the answers but didn't really give specifics, like Fermat's final theorem but for devs), but the fact remains that if you built a modern entity system this way you'd have to rewrite it to get even average performance out of it. No amount of refactoring will fix the base design.

So I feel like I have disagreed directly with his point, and I'm happy to discuss further. I do, weirdly, feel a little like you're not applying the principle of charity to me though. So let's both endeavor to be a little more understanding.

goto11 69 days ago [-]

> I do, weirdly, feel a little like you're not applying the principle of charity to me though.

Fair enough, we can all be guilty of that. What part of your argument have I misunderstood or misrepresented?

camgunz 68 days ago [-]

I think I've pretty directly disagreed with:

> focus on clarity first and only optimize for performance if absolutely necessary

I think that in a lot of very important, very common use cases (take your pick: game engines, embedded programming, OS development, language runtimes, linear algebra libraries, etc. etc. etc.) this is extremely bad advice. It's... *probably* good advice for enterprise applications; but that's a pretty small subset of all software. I think that if you start with clarity here and do performance optimization after the fact you will build something that can't be used as intended--almost always (of course, not "alwaysly", see Python).

More broadly, I think this is a far too narrow way of looking at "clarity" vs. "performance". I have some idea of what I think clarity is, and I have some idea of what I think performance is, and they're both pretty nuanced and deep topics. I think they're two big bags of variables in an even larger bag of engineering variables, and I think Fowler does us all a huge disservice when he skips that entire discussion with what is your synopsis up top there. Maybe you're writing Python and you need a certain level of dynamic nature in order to reach your desired level of flexibility and clarity. That can be fine! The point here is that these are rich topics, and they deserve better than what I see as a pretty blithe dismissal.

Ozzie_osman 70 days ago [-]

Agreed. But that's a fault of content consumers not producers, no? I have read SCIP (in college), Code Complete (first book I got in my first job), Gang of Four (2nd book), Clean Code, etc.

Unfortunately we are in an age where blog posts and videos are a lot more accessible than dense books. That sucks, and means we all have to expend extra energy to find a mix of good and diverse ideas and thoughts.

camgunz 69 days ago [-]

> But that's a fault of content consumers not producers, no?

Yeah, mostly. But at least a little is that Fowler writes and presents pretty authoritatively. And he should, because that's what consultants do, but we should remember that he's a consultant for a very niche area of software ("Enterprise") whenever evaluating his advice.

wst_ 69 days ago [-]

But then, very often, he also make notes that there is no one good design and, when presenting an ideas, he also presents pros and cons of given solution.

vbezhenar 70 days ago [-]

I know one guy who thinks that Fowler is wrong in very large number of points. He works as an architect in big company now (FAANG) and he did some great projects before, so I certainly trust him. For example he thinks that anemic model is the right way to do and putting methods into data objects is wrong. Unfortunately he does not want to write a book yet, so I can't reveal anything, but I just want to underline that a good engineer should not blindly trust anything, even if it seems to be written by a well received author. I, myself, read his Refactoring book and it was great.

Unfortunately I'm not qualified enough to judge those things myself, I just learned that I must doubt about anything, even if it seems to come from big names.

extra_rice 70 days ago [-]

> I just want to underline that a good engineer should not blindly trust anything, even if it seems to be written by a well received author.

Doesn't this also apply to the opinions of your friend who works at a FAANG? How do you judge between theirs and Fowler's view points?

blub 70 days ago [-]

You're completely off-topic. Nobody's mentioning CQRS or anemic models.

This is an article about architecture with zero references to other authors, studies or proof of any kind. It presents several graphs without any supporting evidence.

=> it's absolutely normal to request evidence for those claims. This is software engineering, not software literature, where we try to guess what the author meant and marvel in awe.

It's exactly the younger practitioners on HN which should ignore such articles and demand properly supported texts.

sidlls 70 days ago [-]

Too much of this industry is driven by fads and personalities. You're exactly correct, of course, but I don't think your point will be well received.

Merrill 71 days ago [-]

The Enterprise Architect's job is rather more like that of an Urban Renewal Planner than an architect. There are many pre-existing structures, many interested parties with varying degrees of control, and many economic and technical constraints to be satisfied within a larger political system.

neya 71 days ago [-]

People have asked for evidence on this thread, so let me share my experience. A couple of years ago, I was developing an E-Commerce platformm internally for my company. I follow Martin Fowlers' articles on using DDD

(Domain Driven Design) and Microservices to architect applications. What should have been a month long project took us 8 months! I'm not including the time I took to read about, understand and practice DDD and the surrounding concepts. Every DDD book is atleast 500 pages or more. And none of them are clear on the concepts, and are mostly full of fluff.

To give you more details, our stack was based on Phoenix, but it used a concept called Umbrella applications (basically Micro services pattern). Along the way, I hit so many snags and I thought I must be doing it wrong. Because, that's what these authors kept saying all the time to people who had failed to implement their preachings. So, I met a couple of people who had actually done this DDD on a much larger scale than I had and even they shared the same experience and stories of failure and struggling. Still, I shrugged them off thinking it must be something we were doing wrong instead of DDD/Fowler's teachings itself. In the meantime, we were struggling to launch features for our E-Commerce software because our microservices architecture based on DDD was just a house of cards at this point. And then, I finally called up someone I knew who had done DDD successfully and asked him how he did it. He said one of the authors of a popular DDD related website, had offered to consult for his company. And they had no choice but to employ him because they were pretty invested at this point. After a while, I started meeting a bunch of such companies who got their DDD setup done by either authors or by some vendor these authors had a stake in.

And then BINGO, it all made sense - most of these patterns/shortcuts/discoveries/secret sauces by these authors are hardly tested by them. It's just a way to sell you more consulting and training.

A couple of weeks later, we hosted a casual meetup for tech talks in our area and we got to meet more folks who had followed the same path and failed. We finally decided to do a small hackathon over a weekend and get our projects done. In the end, in about 3 days from Fri-Sun, we had got done so much more with a monolithic architecture than we had ever gotten done with DDD and microservices. Today, my platform is in production and is a proud monolithic app and still going strong.

It was perhaps the most expensive lesson I've ever learnt, but one that I will remember for a long time.

MiyamotoAkira 71 days ago [-]

2015: https://www.martinfowler.com/bliki/MonolithFirst.html

neya 70 days ago [-]

I remember this article very well. When it was published, it was too late as we were already following his articles on DDD before that. This is what I meant when I said the authors said we were doing it wrong. This was published after realizing a lot of people who took these authors' advice and went down the DDD path weren't happy and shot themselves in the foot.

balfirevic 70 days ago [-]

There is some ongoing confusion here. Using domain-driven design has no bearing on microservices vs. monolithic architecture.

sooheon 71 days ago [-]

What I find interesting about this anecdote is that if you give DDD proponents the benefit of charitable interpretation, they may genuinely have had your exact same journey in reverse.

Maybe there is an inversion of the duckling imprinting phenomenon in software, where everyone hates the first terrible architecture they write, and when they finally get it right through lessons learned, preach that as The Way.

meowface 70 days ago [-]

Exactly. That's what this breaks down to a lot of the time. A huge portion of software design, and discussion of it, is a lot more psychological and a lot less empirical or objective than it may appear. That's why you still see people at each other's throats pretty much every day on reddit and HN over pretty much every software design debate ever. (And software usage debates are even worse.) Their method is optimal and their opponent's is terribly overengineered or underengineered.

People like to cling to their preferences, and prefer what they've moved away from when they were less experienced and skilled, or what people they respect use, or what happened to catch their fancy or fascination years ago, or just what makes them subjectively feel good. You see the exact same kind of thing in other disciplines, including artistic fields, and the exact same kind of discussion board vitriol and cargo culting.

The nice thing is you at least get exposed to a lot of perspectives. If someone were to read only 1/4 of the posts in this thread, they may be convinced to take some kind of hardline position and find themselves down a narrow path. But if they read every post, they can take it all in and understand how much vehement disagreement and fuzziness there is, and be more skeptical and critical about the right or wrong way to do things.

bcrosby95 70 days ago [-]

Even Martin Fowler writes that DDD is for large, complex software. He also recommends monoliths before microservices.

If you're writing something that should take you one month it is not large and complex. It is not suitable for DDD or microservices.

Microservices are not always micro. It's a bad name. Teams of 5-10 oftentimes work on only a single microservice. 1 microservice can be millions of lines of code.

neya 70 days ago [-]

See, this is the problem. See, the definition of microservice (in DDD context) is so vague, it keeps varying, but according to books on microservices, they're supposed to be simple and broken down parts of a larger system. Definitely not a million lines of code.

bcrosby95 69 days ago [-]

I think you're reading a lot into those books.

Microservices were pioneered at companies with thousands of engineers. That is where they shine.

Small is relative. The point of a microservice is to allow a smallish team of 5-10 people make their changes independent of all other teams. That is their primary purpose. To make sure your thousands of engineers aren't stepping on each other's toes.

All other benefits range from minor to bullshit, especially if you don't have a huge engineering department.

The above applies even if you don't use DDD. DDD oftentimes goes along with microservices because their bounded contexts are a good way to split your microservices.

flarg 70 days ago [-]

I'm pretty sure DDD was not just Martin's idea and it's still pretty new and evolving even today, and I thought it was pretty standard practice to build a monolith first and the chop into microservices only when you need to. How come you went straight into these things?

gregdoesit 71 days ago [-]

I have been building large scale/distributed systems myself and reviewing designs of systems like these being built around me, for years now. I have never come across any of the type of ivory tower architecture that this post and other sofware architects write about. We're talking about real-world systems that are high-load and somewhat novel, with battle-scarred engineers building them.

I find it ironic that that many of the people who write about architecture are consultants or writers. Consultants usually come into a project, make some decisions and write some code in a few months. Then they leave, without having to maintain the application. Of course they bring in fancy-named architecture patterns as one-size-fits-all solutions, leaving before the cracks on the approach show. Ever notice how little writing there is about iterative software design?

For writers, it seems they build a simple proof-of-concept solution, that is never deployed in production, documenting this process and showing it off as an example.

Yet, when it comes to writing about software architecture, this is 90% of the materials. Engineers who build this day in, day out, design systems from inception, iterate on them, code and deploy, then maintain and operate them for years - they barely share what approach they took. And it's none of this fancy-talk in articles like this one. It's not UML or other formal methods. It's simple documents, simple diagrams, lots of discussion about tradeoffs and tons of back-and-forth comments over Google Docs/O365 and in real life.

I'm at the point where I'm seriously debating writing something more substantial on how *actual* software design works at tech companies, when you're not a consultant or a writer, but an engineer who does this full-time, end-to-end, living with all the consequences of decisions and fixing mistakes after.

On the importance of prototyping, extensible MVPs, whiteboarding, documenting an initial design and getting feedback, iterating on it, architecture jams, design war rooms - and when starting again when the world/business changes substantially, keeping things running and migrating things over to the new system.

tootie 71 days ago [-]

I've worked in both a big IT orgs and consulting and the Ivory Tower Architecture people are 10X more prevalent in IT orgs. They're the ones who establish architecture review boards and request extensive documentation and rounds of approval before allowing a project to proceed. On the consulting side, we typically define an approach which might as simple as saying "We're going to write microservices, mostly in Python and our main data store will be Redis until we pick something better" and then start iterating.

wolco 71 days ago [-]

When consulting I love those guys. So many billable hours and project delays with pay.

jammygit 70 days ago [-]

> I'm at the point where I'm serioulsly debating writing something more substantial on how actual software design works at tech companies

If you need a little push to do so... I dare you

hliyan 71 days ago [-]

I've been doing exactly the same for the past 16 years, and my experience has been exactly the same as yours. Never seen any of these proposed models work in reality, and never seen these models come from industry or academic veterans.

notus 71 days ago [-]

I work at a fairly large company you've heard of and there is definitely a lot of application of the practices that he preaches. It never works perfectly because nothing ever does, but the effect of his teachings is noticeable. They are ideals to strive for. The failure at implementations tends to be more a people and process problem than a technical problem.

wincent 71 days ago [-]

"I'm at the point where I'm serioulsly debating writing something more substantial on how actual software design works"

It's probably already been written; see the "Big Ball of Mud": http://www.laputan.org/mud/mud.html

jammygit 70 days ago [-]

> While much attention has been focused on high-level software architectural patterns, what is, in effect, the de-facto standard software architecture is seldom discussed. This paper examines this most frequently deployed of software architectures: the BIG BALL OF MUD. A BIG BALL OF MUD is a casually, even haphazardly, structured system. Its organization, if one can call it that, is dictated more by expediency than design. Yet, its enduring popularity cannot merely be indicative of a general disregard for architecture.

>These patterns explore the forces that encourage the emergence of a BIG BALL OF MUD, and the undeniable effectiveness of this approach to software architecture. What are the people who build them doing right? If more high-minded architectural approaches are to compete, we must understand what the forces that lead to a BIG BALL OF MUD are, and examine alternative ways to resolve them.

Interesting!

mrkeen 71 days ago [-]

Would you expect your writing to be received any differently?

alanfranz 71 days ago [-]

Martin always has good points. What I miss from him is proofs. The weeks vs months part crossover for internal quality... How is it measured? Is it just a thought, or a wish?

moksly 71 days ago [-]

In the public sector of Denmark we've been on a decade long journey toward better Enterprise Architecture. We have national guidelines for how we want public system/services to be build and how they should be able to transfer data through APIs. We don't get completely technical, telling you what tools you need to use to build your software, only how your software needs to fit into the environment that is public sector digitalisation.

The reason we do this is because we've seen what happens when we don't.

The municipality where I work operate a pretty common amount of IT systems. We don't know exactly how many because there isn't a centralised unit to handle them all, but our guess is 300 different IT systems. There is a lot of common things these systems need, like authentication. That used to be terrible. Before web-services and federated IDPs it was quite literally maintaining our organisational hierarchy in 300 different IT systems manually. And that's just our place, in Denmark we have 98 municipalities that all have 300 IT systems run by different suppliers. With a common architecture we defined how organisational data needs to be interpreted by these systems, we also build a single co-owner database where each municipality could make these data available in the same defined format, through the same APIs.

It's already saving us millions, just from this tiny piece of the much bigger picture. It's not been without backlash either, because as it turns out, private tech companies don't actually want to build the smartest systems. They want to sell us silo systems, because then they can sell us access to our data 300 x, where 1x could be organisation data and another x could be statistics.

Anyway, I think we have evidence that architectural choices matter a great deal. At least when you look at it from an enterprise sized point of view.

With that said, I'm not sure I buy detailed application architecture, where you design everything in your favourite markup-language, either. We certainly have no evidence to support that this sort of architecture isn't just a waste of time. But for the big picture, in enterprise sized organisations, you can't live without IT-architecture if you want to be efficient.

If you want to read more about it, you can at: http://info.rammearkitektur.dk/index.php/Forside but it's in Danish.

peheje 71 days ago [-]

It sounds more like someone actually sat down and thought about what to do before doing it.

I read danish. I have experience with it. I did not understand what the page was trying to say.

Providing your data through APIs are often a better way than rolling your own manual delta sync through reports every Sunday uploaded to your company FTP server with 5% downtime.

Not duplicating your data throughout various systems is not architecture it's just common sense.

moksly 70 days ago [-]

> Not duplicating your data throughout various systems is not architecture it's just common sense.

I wish you were right about that, I really do. Unfortunately only 7 of our 300 IT systems do this today. As rammearkitektur compliance is now a required part of public procurements that number is going to go up, but only because we're forcing this architecture upon the private tech houses.

If we didn't we would never see 95% the APIs we so desperately need, and the last 5% would cost us half a million each. That's how it's been for 30 years, and it's only changing because of enterprise architecture.

PeterisP 70 days ago [-]

Look from this from a perspective of a vendor. If a municipality orders development of a new system that does X, then from the perspective of a vendor, it's much cheaper to plan a 'fresh' system that has all the data to do X (and nothing else), structured in a way that makes it simple to do X.

If you want to actually understand all the other municipality systems that hold that data and integrate with them (instead of duplicating the data), then your proposal is not going to be the winning lowest-cost bid.

Spearchucker 71 days ago [-]

Oh man. Another diatribe from Martin Fowler™.

Takes me back to the 90's when I was in awe of teh architectures and architects, understanding of which was always just beyond my grasp.

99% of software I've encountered has some kind of interface, a domain it operates in, and data it operates on. Call these three things layers. It also has some cross-cutting things that are universally available in the software - broadly, security, comms and exception management.

You deploy these things on desktops, phones, and devices. Other parts of the things live on servers or on clusters, mabe. Call these places (hardware) tiers.

There. Architecture done. From micro services to huge banking monoliths. Start there. Everything else is the usual let's think about what we're doing before we do it.

But I guess keeping it simple and dealing with exceptions as they arise isn't as glamorous as using fancy new names, working for a hipster consultancy or generally saying X or Y is teh new futures.

If you harbour doubt great, questions are good. I suggest obtaining a copy of Wicked Problems, Rightous Solutions (ISBN-10: 013590126X). It will show you that we had EXACTLY the same problems of today, back in 1990. We just call them new things.

wpietri 71 days ago [-]

You say this with a lot of bile, but I'm not sure where your opinions actually differ from Fowler's. He says right in the second paragraph that he's suspicious of the term "architecture" because it "often suggests a separation from programming and an unhealthy dose of pomposity."

I think keeping it simple and dealing with exceptions is great. And so does Fowler, I'm sure. But I've been doing that a long time and I've learned some lessons about what "simple" means, and how some ways of dealing with exceptions work well in certain circumstances, while others can seem appealing but end up being a mess.

Like you, I think high-church, prescriptive software is mainly a giant waste, a way for fast talkers to get big paychecks while performing oh-so-smartness. But that doesn't mean over decades of coding we can't learn anything about they ways we build systems.

lemmsjid 71 days ago [-]

Having similarly done software in that time period, your reductionist argument flies in the face of my own experience, where there are actually tons of nuance within what you are correctly identifying as typical macro patterns. When I look at Fowler's repository of EA patterns (https://martinfowler.com/eaaCatalog/), I see a ton of time tested patterns that are, to this day, used in a tremendous amount of software. They are so prevalent that they are typically abstracted into frameworks or libraries that are then used by people building line of business applications.

Something I have always appreciated about Fowler has been his workman-like, semi-anthropological approach to documenting these patterns. He does not claim ownership to them, but instead discovers and names them out of working software. He does have a playful opinionated-ness to his writing, but that is really only to spice things up, as far as I've seen (in other words, he has been quite willing to modify his own positions over time).

Perhaps to your point, I have seen people read these books and suddenly wish to apply all the patterns at all the times to all the software, resulting in an ill conceived amount of complexity. That, to me, is why a healthy software development team needs senior developers who have been around the block on this sort of thing. And why the agism in the industry is so poisonous and counterproductive. A healthy team needs healthy defenses against the forces of over complexity (and, equally, over simplicity), and because of the tremendous amount of abstraction in software engineering, experience, erudition and mentorship can bring that to a team.

stephen 71 days ago [-]

Fancy new names is basically the antithesis of Fowler's approach (it's driving shared language/canonical definitions).

And personally in the mid-2000s I found Fowler's material to, contrary to your assertions/experience, actually be a source of simplicity (by at least explaining pros/cons/alternatives in a comprehensive/vendor/platform agnostic manner) vs the other "just run this huge EJB server, np!" approaches there were mainstream at the time (and personally who I think would be a better target of your ire than Fowler and ThoughtWorks).

camgunz 70 days ago [-]

I have what I feel like are cultural disagreements with Fowler (e.g. "Enterprise Architecture" is oxymoronic to me) so adjust for my bias accordingly. But perhaps his most famous book is "Refactoring", where he gives (fancy, new) names to things he found in the wild. It's basically what he does and arguably his most influential contribution to the industry.

tootie 71 days ago [-]

As far as I understand Fowler's posts, it's mostly about distilling best practice and ways of thinking. I haven't seen anything advocating anything more complicated than what you're describing. I've seen a lot of enterprises that don't do any of this. Ball of Mud is still pretty prevalent.

jsdalton 71 days ago [-]

Is that 99% of software all the same? Like, have you really not observed huge differences in the readability of the code you encountered? Have you not found that sometimes it is painful (or close to impossible) to extend or build more features on top of a particular system, while other times it is quite straightforward, even a joy?

If your answer to these questions is yes, to what do you attribute the difference? How would you counsel a young software engineer or team? How can we write better, more maintainable software that still allows us to deliver features quickly?

scarface74 71 days ago [-]

I learned a lot from Domain Driven Design and some from Design Patterns by the GOF.

jeffdavis 71 days ago [-]

Architecture is your chance to decide which problems will be easy and which problems will be hard.

In a microservice architecture, failure handling is easy. State management is hard.

In a centralized database architecture, failure handling is hard. State management is easy.

ris 71 days ago [-]

> In a microservice architecture, failure handling is easy

I'm not sure I would agree with this - it's still incredibly hard to be able to make the rest of your architecture do something useful when a significant part of it goes away. Multiply the possibility of one of your important services going down by the number of important services you have...

Kortaggio 71 days ago [-]

The author's statement that "My view is that applications are a social construction" I think hints at a deeper truth, that while _code_ is run by machines, managing a _codebase_ is more like raising a plant in a garden and not like changing the oil in a machine. Reminds me of "A Codebase is An Organism"[1].

[1] https://meltingasphalt.com/a-codebase-is-an-organism/

dredmorbius 71 days ago [-]

See also: Conway's Law.

http://melconway.com/Home/Conways_Law.html

blub 71 days ago [-]

Whenever I see yet another article by Martin Fowler, Kent Beck, Robert Martin & co I ask myself - where is the evidence for what they are preaching? What are the graphs based on? What is the evidence behind the proposed rules?

These authors are clearly accomplished blog and book writers, but anyone whose *software-related* accomplishments are not open source or at least well known should have their statements scrutinized more closely.

I don't see any studies cited, case studies presented or concrete examples. All I see is yet another reasonably sounding yet unsupported piece of folklore.

chrisseaton 71 days ago [-]

For some of these speakers, I've literally never seen any evidence that they've written a single line of code. Why should we listen to Fowler? What large successful projects has he created using his ideas are there that I can look at? Where's the data behind his slides? It's all just talk talk talk.

I'd listen to someone like Chris Lattner talk about how to design and write software because I can see whatever approach he uses must work well. I should listen to these full-time speakers because... I don't really know and perhaps we should start challenging their talks by asking for the data behind them.

rhizome31 71 days ago [-]

I don't think Fowler pretends to be much of an innovator. The way I see it his work consists in collecting and documenting existing ideas from the industry. You'll find many instances of patterns listed in PoEAA in open-source frameworks. His "refactorings" are also pretty simple ideas that people generally come up with on their own after a bit of work experience. It's just nice to have a nicely formatted book where they're given a name and they're illustrated by diagrams and examples, especially to help junior developers. You'll also notice that for each pattern or refactoring, Fowler generally tries to show its caveats. So it's not like he tries to sell *his* ideas at all costs. Note that here I'm talking specifically about Fowler, whose work I have found helpful with my real world projects.

scarface74 71 days ago [-]

Chris Lattner knows how to write system software and compilers which are completely separate beasts from the typical bespoke enterprise or software as a service apps that most developers write every day.

vfc1 71 days ago [-]

Exactly, many times these famous authors advocate for practices that make sense for building compilers and frameworks or libraries, but that make little sense for building applications.

Building applications is very different, because an application is the very end of the software tree dependency. Some practices make sense for both types of software, but many others don't and some of these authors don't seem to be able to make the difference.

scarface74 71 days ago [-]

And sometimes it makes sense to take shortcuts that don't make sense from a long term maintenance standpoint just to survive long enough to get product market fit. Get funding and survive.

Twitter is a perfect example of this. Everyone agrees that Twitter made some bad architectural decisions that caused the frequent appearance of the "fail whale". But eventually, they got enough funding via combination of the VC funding and later the public market to rearchitect their system.

legulere 71 days ago [-]

System Software isn't all that different from enterprise software. The business part is simply of technical nature. The biggest difference usually is that the software developers don't have a deep understanding of the business rules.

scarface74 71 days ago [-]

I’ve had to maintain a bespoke compiler/IDE/VM for Windows Mobile/CE before and have spent two decades writing business software and architecting around the entire stack from networks, to databases, to web servers, load balancers, monitoring tools etc. I’ve also written SOWs, Done project plans, led teams (hated every minute of it). I’ve seen both sides.

The types of optimizations and thought processes around enterprise software is completely different than the optimizations around compilers.

The “entire stack” with enterprise software is much different than the “entire stack” with the compiler. I haven’t had a need to look at the assembly created from my compiler as a business software developer since around 2002.

For context, my first hobby programming was in the 80s optimizing 65C02 programs by using zero page memory access to eke a little more speed out of the 1Mhz processor and trying to use branch statements instead of JMP statements because they were faster.

legulere 71 days ago [-]

Of course you’re not looking at generated assembly anymore because that is the business value of a compiler. The biggest difference now is that you’re probably not as big an expert in the enterprise topic as you were about generating assembly.

scarface74 71 days ago [-]

These days - no. I’m nowhere near the business expert in the healthcare industry where I now work as I was in the ins and outs of assembly. That’s not my role. I’m a software developer by title, but I would consider myself an “expert” in both the software architecture and the AWS infrastructure of our (smallish) company.

When I was working for the company where I did have to maintain the compiler/IDE/VM after they forced their founder out. I also had to know the complete business needs of the client that kept us afloat because as they laid off people, I had to interact with the customer and write the SOWs. I saw both sides. The skillset I needed to implement the business side was completely different than the skillset I had to have to maintain the compiler stack.

Luckily I was mature enough (and an MBA dropout) to understand the business end and enough of a geek to understand low level computer concepts and could read assembly.

EdwardDiego 71 days ago [-]

https://github.com/unclebob might be a good place for you to start. Fitnesse is probably the biggest one on there.

You can't really hold the fact that most of his working life has been for closed source orgs against him.

streetcat1 71 days ago [-]

So first, you should read their books for concrete code. Martin books, for example "refactoring" or "Enterprise architecture" actually have good code examples.

The same apply for kent beck early work on smalltalk (you can look at Junit) or his early work on the HotDraw editor (which was the source of many of the pattern in the Design Pattern book). Kent was one of the first adopter of smalltalk at the early 80 is techtronix, together with ward cunnighum (the inventor of wiki) and rebbeca swift brooks.

Robert Martin work is mainly driven by his consulting practice and tend to change based on the on going trend. At the 2000's he was working with Grady Booch, preaching UML and design before coding. Later he dropped that in favour of agile and TDD.

You do not see case studies since the industry simply does not really care. It is very trend prone.

Most of the adopted advances were actually "hacks". For example C++ was invented out of necessity to create "C with objects" by a single person in AT&T labs. Unix was hacked in 3 weeks. Java was invented to program toasters, etc.

Any attempt to develop tools and technique based on real research saw not adaptation - for example Ada at the 80s, UML at the 90's and early 2000, functional programming, TLA+ , etc.

geofft 71 days ago [-]

The question was about "research" in the meaning of quantitative scientific evidence that these proposed techniques make software delivery better in real-world conditions, not "research" in the sense of new ideas from academics. As a professional, I don't want UML because I've seen no evidence it actually helps. Show me relevant studies that it does and I will absolutely change my mind.

(And there are plenty of things that industry does respect that came from academic research in the sense of new ideas. Rust, stereotypically the language of the trend-prone, draws on everything from Ada to functional programming. But unlike those it actually and demonstrably meets the needs of general-purpose programming in industry.)

streetcat1 71 days ago [-]

Sure. So UML is based on two main research areas - UML class diagram is based on rational algebra theory. A correct UML class diagram should follow the same normalization roles as defined by Codd seminal paper - "A Relational Model of Data for Large Shared Data Banks".

You can ask, how does Prof Codd knows that this is is the correct way to describe and implement data. Well this follow directly from set theory. Again, how do we know that set theory works - well here we are the level of axioms of the set algebra.

BTW, this Codd work was also "hacked" again by system R research which invented SQL.

The other part of UML is based on finite state machine, which originated by Turning, and later refined by the work of David Harel on state charts. Again, those are theoretical CS tools. However, the work of Harel on state charts, was originated from the work on the Israel first aircarft (to be cancelled later), where he needed to define the the missile system.

If you are interested on how those tool originated and later combined see the work of David Hay - "UML-Data-Modeling-Reconciliation"

Those tools are useful in scale. For example, system with 3000 different use cases. Or for example you have 200 - 500 classes, each is a state machine.

Again, since there is no software professional standard (not software engineering bar exam), you can or cannot use anything.

geofft 71 days ago [-]

That's, again, not what I'm asking. I'm not asking for the theoretical underpinnings of why this is a good way to represent 200-500 classes. I'm asking why, if I and two coworkers need to code up 200-500 classes by next month to meet a promise to a customer, spending time on UML will make the customer happier. I have a programming language that itself has strong theoretical underpinnings. I also have a pretty good hunch that in about two weeks there will be a last-minute requirement that changes the structure of the data model, and in about three weeks there will be a production outage of another system that distracts my attention.

Will UML help me deliver software with changing requirements faster and more accurately? How does it compare to other options in my toolbox, like using a less verbose language that doesn't have real OO (which in my experience is often the right choice) and because of its conciseness is easier to make major changes to? How do I tie the UML object structure to my programming language, and why is it better than using a strongly- and richly-typed language in the first place? Why do I see many successful software projects that as far as I can tell don't use UML?

streetcat1 71 days ago [-]

First, I must say that the inventors of UML saw it as the last layer. The grand vision was a complete code generation from UML diagrams. And this was the overall grand vision that drove OO in general. I think that this is was happening now with the "low code" startups.

The whole idea is to separate the global decisions (which are hard to change) - e.g. architecture, what classes, what each class do, from the local one (e.g. which data structure to use). So you would use UML for the global decisions, and than make programming the classes almost mechanical.

Of course the customer do not see that. The customer see the user interface, and if the overall use cases are implemented correctly. Hence using those tool might help or not.

Lets take architecture for example, why do people use an architect ?. Why does architect needs diagrams? Will the owner care of the architect use diagrams?

geofft 71 days ago [-]

Again, I am not asking how to use UML, or why the inventors of UML designed it in a certain way, or why it sounds like it could be a good idea, or whether there are other systems like UML that are good ideas (I can just use those systems directly then!).

I am asking what evidence there is that UML helps me deliver better products more efficiently. Are there successful software projects that do this complete code generation thing (and how do they compare to directly writing real code in more efficient programming languages with rich typesystems)? Are there development efforts where giving up and deciding that certain architectural elements are hard to change has been the right approach (in my experience it generally never is)? What are their names? Do they have experience reports? Are there numbers?

I am looking for, at the least, a specific name like "The Chrysler Comprehensive Compensation System took these approaches to development." (But hopefully not a massive and expensive failure like C3.)

streetcat1 71 days ago [-]

So you would not find any. You would need to judge those tools based on your specific use cases.

Even if you find such and such project did X and got Y, the sample size is too small.

The issue with software is that each project is different, and hence we cannot generalise from any given project.

Jare 70 days ago [-]

> The whole idea is to separate the global decisions (which are hard to change) - e.g. architecture, what classes, what each class do, from the local one (e.g. which data structure to use)

And this would be one of the reasons why UML is objectively a terrible idea for developing great software efficiently. (I assume it is ok for developing poor software at great cost)

netsensei 71 days ago [-]

I think the answer to your question would be: it depends on external constraints. How much time do you have? What is the scope of the project? Size of your team? Number of stakeholders? Budget? Etc.

In my mind, the crux of the entire exercise is trying to translate a complex murky real world problem domain into a mental model that can be expressed via a shared language describing properties, aspects, processes, concepts, etc.

The challenge is getting that translation done in a timely, secure, efficient, performant, maintainable manner.

Quantifying that translation into a comparable measure? That's where you'll find your in the same boat with many other project managers who make guestimates and hope that everything pans out in the end.

Trying to see it as a purely computational problem will confront you with that other evergreen: the P equals NP problem.

My point is that trying to find a hard model that answers your particular question pertaining to UML is very hard since there is no neat general foundational model here that allows you to compute hard, quantifiable predictions about the usefulness of a given tool.

I suppose it's like with so many trades. It depends on the experience of the expert to assess what strategy is best. It's okay to question that experience in hopes of avoiding all too human biases that cloud our view from what we are actually trying to solve. Even so, questioning any and all statements because you can't boil it down to hard data equally might threaten to narrows one's view.

If your experience tells you that UML works for you as a tool, then, by all means, stick to UML. Even when there is more then one approach to a solution. If you use pen and paper to quickly draw stuff, well, that's equally valid if that works for you. I just wouldn't dismiss an approach because its usefulness can't be measured up front. Reality is far too complex and sometimes there's simply no point in dwelling too much upon all of this in the first place.

ori_b 71 days ago [-]

> *I think the answer to your question would be*

Yes, that's the problem. The pixels spent here are based on hypothesis, not experiments. You think one thing, I think something else, and not a joule of effort is spent measuring.

netsensei 71 days ago [-]

Ah yes, that would be Karl Popper's concept of Falsifiability of a hypothesis.

https://en.m.wikipedia.org/wiki/Falsifiability

That's what we are debating here: is this claim that use of UML is always markedly better when modelling any problem domain is true as observed through a sufficient number of data points?

I'm poking holes in that hypothesis because there's always n + 1 conceivable use case where the opposite is true. And I'm basing myself on set theory and computational algebra.

The crux is 'sufficient' because all use cases that could be moddeled with UML being equal, prove to me that there is a finite amount of use cases. Hard answer: you can't.

Moreover, there is no way you can prove to me that each and any use case is solvable with UML in an efficient way without actually solving said use case.

Check Hilbert's Grand Hotel paradox. Perfectly applicable to this question.
https://en.m.wikipedia.org/wiki/Hilbert%27s_paradox_of_the_G...

ori_b 70 days ago [-]

> I'm poking holes in that hypothesis because there's always n + 1 conceivable use case where the opposite is true.

And it's conceivable that the sun will not rise tomorrow, or that the law of gravity will pass an inflection point and reverse itself. Also based on "set theory and computational algebra", and the impossibly of testing each possibly.

What's your plan for tomorrow, and does it depend on gravity?

netsensei 70 days ago [-]

I'll take logical fallacies and rethorical devices for 200, Alex.

ori_b 70 days ago [-]

And what logical fallacy was Hume committing when he explored the limits of inductive reasoning with that example, precisely?

traderjane 71 days ago [-]

geoftt isn’t asking whether metaphorical TLA+ is academically fruitful. He’s asking if Amazons et al. have _continued_ to use it, and whether there’s some volume of evidence attesting to this.

streetcat1 71 days ago [-]

Sure. So if amazon did not continue to use it. Does TLA+ suddenly become not useful?

My point here is that those tools are theoretical CS tools based on the underlying nature of computers - which are a discrete digital state machine.

Why do you even use high level languages, why not just program in assembler?

traderjane 71 days ago [-]

If the Amazons of the world tried and dropped TLA+, that is signal to people who were on edge of technological adoption. If we received data on its efficacy during its trials, we'd be even more informed, as people are interested in more than the internal consistency of a system.

Right now some people have a theory that Uncle Bob is noise in the sea of architectural opinion, and we can't move forward because we don't have enough data. The call for more data is welcome by me.

spenczar5 70 days ago [-]

Amazon continues to use TLA+. I work there. It's not a tool needed every day, and it's not needed by everyone (really, not by almost _anyone_). But sometimes it's the best tool for the job.

You won't be getting much data on this because companies are secretive.

The way it's used is to ensure correctness of distributed systems. A classic example might be ensuring that a model for backing up customer data is unable to get into indeterminate state. It's important to get the model right. But the model isn't code - TLA is not a programming language, it's a tool for modeling state machines; yes, of course, one needs to also write software to deliver value.

geofft 71 days ago [-]

> Sure. So if amazon did not continue to use it. Does TLA+ suddenly become not useful?

Yes. (Or more precisely, it never was useful.)

Zero part of my business requirements involve anything about discrete digital state machines. If I could deliver business value more efficiently with pen and paper, I would. If I could deliver business value more efficiently with Excel or a shell one-liner, I would (and do). If I could deliver business value more efficiently by singing my harmony into the Music of the Ainur, I would. I write software because it's the tool we've found that is most efficient at delivering business value, not because the software itself has value.

So I use high-level languages because they are demonstrably more efficient. (And I frequently write sloppy code in Python or awk because it helps me answer business-relevant questions like "why is the site slow" quickly, without demanding I be rigorous about my software engineering in the process of answering the question.) don't use UML because I have yet to see evidence that it will help me deliver business value more effectively.

(I do also enjoy writing software as a practice, and I do that on weekends. That's the time to make code for code's sake.)

jhayward 71 days ago [-]

I would be interested in a list of the tools you use daily and your list of research publications that prove that those specific tools improve productivity, reliability, or whatever metric you have named in selecting them.

geofft 71 days ago [-]

That's a fair question, and it is absolutely fair and correct for you to imply that I don't have any.

However, the comment above (which was not me) said "studies cited, case studies presented or concrete examples," not just studies. I was taking objection to the claim that merely coming out of academia counts as a "study" in this sense. I don't need an experimental study (though I would love to have one), a body of anecdotes of successful uses of a tool is sufficient.

I can name countless case studies, concrete examples, and anecdotes of people using the programming languages and software environments I use successfully. There is a large body of evidence that, say, Linux works fine, whether or not a paper says as much. (And, in particular, there is a sizable body of anecdotal evidence that OpenStack does *not* work fine, and I've been pushing my team to find a migration strategy out of OpenStack for this exact reason.)

Also, I think the burden of proof is on someone advocating for change. When I am delivering software successfully, and someone else comes in and says "You need to be taking this approach," I think it's fair for me to say "Why?", and that that's very different from staring blankly at bash and saying "Why?" and refusing to type anything.

And finally I would actually be delighted to have real metrics for my tools. I do want to know if, say, enforcing style checks via pre-commit hooks makes development velocity faster and avoids annoying developers! It certainly seems like it would but it sounds like an easy enough thing to test, and I wish we as an industry (and more specifically people in either academia or industry who see their work as advocating for better software development practices, not developing software) would spend the time to test it.

chrisseaton 71 days ago [-]

I can see 'before' and 'after' code in their books.

How do I know the 'after' code is any better in practice?

streetcat1 71 days ago [-]

I think that the main issue here is to define what is better. For example. If I am creating a new startup and have money for only two months - "better" in this case is rushed out code (whatever that means). I.e. the software have no future in any case, if it is not delivered on time.

However, if I develop radiology machine. Better in this case is very exact state machine description, otherwise I can kill patients.

In those author specific case, we are talking about business system in the enterprise. The "better" here is how fast and easy I can change those system, since I get new requirements every week.

rhizome31 71 days ago [-]

You use your best judgement based on the context, as you usually do when developing software. The only thing you've gained by applying a documented pattern is a bit of help to write your commit message :-)

chrisseaton 71 days ago [-]

> You use your best judgement based on the context

This isn't how science is supposed to work.

I work on programming language performance - when I say something could be implemented in a faster way, people don't just say 'yeah using my best judgement that'll be faster', they ask me to prove it. Otherwise I'm laughed off the stage. Why is it different for these people?

rhizome31 71 days ago [-]

I might be wrong, but did Fowler pretend to follow the scientific method?

I work on developing business web applications and Fowler's work has been very helpful to me, especially when I was a beginner. I still sometimes refer to PoEAA when picking up a framework that implements a pattern described there. However I'd be surprised if Fowler had anything useful to teach to someone working in your field.

chrisseaton 71 days ago [-]

> did Fowler pretend to follow the scientific method

His company calls him a 'scientist' - a 'chief scientist' no less - https://www.thoughtworks.com/profiles/martin-fowler and presumably bills accordingly.

rhizome31 71 days ago [-]

Oh I see, that looks like marketing BS indeed. Not cool that he agreed with this job title. But to be fair, these days you also see a lot of 'data scientists' whose 'scientific' work consists in putting some statistics together and plot them with Python.

Anyway what I was saying was based on reading his books, which again I think have real-world value for some areas of software development. As I said in another comment to me his work consists in collecting and documenting existing practices. I don't think he pretends to teach magic recipes. Each technique is usually presented with potential drawbacks and context where it might apply. That's why I was telling you to use your judgement because these techniques are not supposed to be one-size-fit-all silver bullets. The point is more to identify and describe recurring patterns so that we can communicate more clearly, hence my light-hearted joke about how it helps writing your commit messages.

I agree that it would be great to have strong scientific studies showing that such practice works better than such other practice and I bet Fowler would agree too. In fact I watched a talk recently where he advertised a book called "Accelerate" (not by himself) where apparently the authors attempted to evaluate the impact of practices associated with DevOps based on statistical data.

The thing is it's much easier to demonstrate or measure the performance of an algorithm than the impact of a code change on the whole life-cycle of a software project. So we resort to experience and intuition not because we don't value the scientific method, but because the scientific method is too difficult to apply in this context.

blub 70 days ago [-]

I read parts of the refactoring book (1st edition) when it came out to see what kind of refactorings the author mentions. Since I didn't remember much from it, I decided this year to take a look at the 2nd edition, which to my amusement is written in JavaScript. After reading the example refactoring at the beginning I abandoned the book, because it champions the "code as prose" idea, which is rightfully contradicted by Ousterhout in his "A Philosophy of Software Design".

"Enterprise architecture" was not relevant for me, I wasn't doing any EA when I started reading it and quickly lost interest. The only potentially engaging topic was concurrency, but there are much better books available about that topic.

"Pattern-Oriented Software Architecture" (I am only familiar with vol 1 & 2) has wider applicability and is more compelling in my opinion. And they have references and examples. For instance the Pipes & Filters architecture patterns gives UNIX [Bac86], CMS Pipelines [HRV95] and LASSPTools [Set95] as examples and references "The Pipeline Design Pattern" [VBT95].

I do see case studies and references, but not from these authors I mentioned.

manojlds 71 days ago [-]

I used to work at ThoughtWorks and Martin Fowler's inputs are based on his interactions with many many ThoughtWorks' project teams.

When he visited my team for one of those learnings discussions, I was amazed with the way he was able to absorb and present the same things we were learning in a much different and easier to understand manner.

millerm 71 days ago [-]

I have had this thought for many years. Where is all the perfectly designed, bug free, maintenance-bliss, fully documented, fully tested, future-proofed code located so we can all marvel at its glory?

mirceal 71 days ago [-]

> All code is bad. Every programmer occasionally, when nobody’s home, turns off the lights, pours a glass of scotch, puts on some light German electronica, and opens up a file on their computer. It’s a different file for every programmer. Sometimes they wrote it, sometimes they found it and knew they had to save it. They read over the lines, and weep at their beauty, then the tears turn bitter as they remember the rest of the files and the inevitable collapse of all that is good and true in the world.

from: https://www.stilldrinking.org/programming-sucks

michael_j_ward 71 days ago [-]

I haven't gone through this yet- but my understanding is that there are some examples of this here "The Architecture of Open Source Applications" [1].

[1] http://aosabook.org/en/index.html

mobjack 71 days ago [-]

You are comparing the perfect against the good while Fowler is likely comparing the good against the bad.

Taking an extra week or two to clean up bad design decisions early in a project can pay dividends in the future.

Spending months extra trying to do the perfect architecture is harder to justify.

pjmorris 71 days ago [-]

To take the other side of this coin, where is the evidence that evidence makes a difference? At least, where is the evidence that evidence is what convinces a software developer or manager to change something about their process?

There are entire conferences [0,1] where hundreds of researchers present peer-reviewed studies, case studies, and concrete examples, evidence for various software engineering claims. Thousands of papers have been published at these conferences over time. Usually when those papers are linked here, they don't get much attention, e.g. [2], [3], [4].

As a rule, industry doesn't pay attention to them, except to hire the grad students out of academia. At these conferences, they spend a lot of time discussing how to get more industry participation. But the things they try haven't really worked so far.

To go to all the trouble of generating peer-reviewed science only to have it ignored may not be the best trade available for how Fowler, Beck, et al spend their time. There are doubts about whether it's the best way for academics to spend their time.

[0] http://www.icse-conferences.org/

[1] https://www.esec-fse.org/

[2] https://news.ycombinator.com/item?id=20064211

[3] https://news.ycombinator.com/item?id=20575415

[4] https://news.ycombinator.com/item?id=19714085

Chris_Newton 71 days ago [-]

There's nothing wrong with just suggesting reasonable ideas that might be useful. If everything needs to have a solid evidence base or rigorous supporting arguments before we talk about it, it becomes too easy to shut discussion down prematurely without ever discovering whether an idea does have value for someone, particularly in an industry like ours where much of what we do is hard to measure objectively.

It's helpful to be clear about whether an idea being presented is just someone's personal thoughts or anecdotal experience or whether it does have a stronger foundation. If something is presented as a firm recommendation of good practice then it's fair to expect the latter. But it's also dangerous if we reflexively disregard any idea that doesn't already have robust evidence behind it.

jmartinpetersen 71 days ago [-]

Robert Martin seems to cherry pick stats and trends and whatever that fits his world view. In Clean Architecture he cites a study with N=1 as proof TDD works. I have no hope for him.

sooheon 71 days ago [-]

I'm genuinely curious what the stats backed world views you hold are.

billfruit 71 days ago [-]

I generally dislike building arguments on top of statistics, rather than on theory which to my eye almost always looks more convincing. With statistics it is as if the burden is passed on to the reader to check the veracity and validity of the said numbers, where as if it is an argument constructed out of theory, the whole of it is present there for the reader to judge intrinsically on its own merits.

gunnihinn 71 days ago [-]

That restricts you to mathematics, logic or philosophy. Engineering, or the rest of the world, operates _at best_ on statistics.

billfruit 71 days ago [-]

Programming in general and Software correctness does not depend upon statistical arguments, so perhaps such a view could be taken for software engineering too.

Also even considering aesthetics and elegance, I find use of statistics to power a reasoning, ungainly, compared with other means of arguing the same thing.

crimsonalucard 71 days ago [-]

You're right. Why are you voted down. Programming doesn't need to be based on statistics it's largely a deterministic system amenable to proof and logic.

This application of "engineering" onto programming is largely the result of people not truly understanding the system they are dealing with. The system isn't some sub-sonic jet that will be subject to unknown turbulence, it's just axioms and rules.

meowface 70 days ago [-]

Ideally you'd have some of both.

randomcarbloke 71 days ago [-]

Thank you.

If I want to design excellent architecture I will crack open SICP and transpose it to the fabric of modern computing.

I also take issue with much of what Fowler et al say, but that's another conversation.

charlieflowers 70 days ago [-]

I think one of the notions behind the Fowler criticism is something like this: "Early on, a bunch of people got famous as programming thought leaders, but later we all found out there were much deeper, smarter thinkers who didn't get all the buzz."

So the "early" ones might be Fowler, Uncle Bob, Beck, Grady Booch, Evans (DDD), etc.

And the "deeper" ones would be the authors of SICP, or Peter Norvig, or Chris Lattimer.

I can see where this sentiment is coming from, and there's no doubt truth to it.

I'd say SICP is _without a doubt_ deeper and more insightful than most or maybe all of what Fowler has written. From that standpoint, SICP should be more famous than Fowler, but it doesn't seem to be the case. It definitely was not the case 10 years ago.

Still, let's not throw the baby out with the bathwater. Fowler's writing has explained some difficult concepts very clearly. And he chooses topics wisely (but not perfectly in hindsight).

So yes, there are two "tiers" of "experts" about building software. Clearly there is a "tier 1" that is better, more proven, deeper, than Fowler/Martin/et al.

That puts Fowler and some others at tier 2 (or lower, all subjective of course).

That doesn't mean tier 2 is entirely worthless though. IMO far from it.

notus 71 days ago [-]

SICP isn't going to teach you excellent architecture for large systems, it's going to teach you excellent architecture for small programs.

crimsonalucard 71 days ago [-]

The design of small programs is isomorphic to the design of large programs at all levels.

You're just referring to real world caveats and details, but the overall concepts taught in SICP apply to all layers of programming from assembly to the architectural level.

Small programs:

```
Q =  A -> B
W =  B -> C
R =  C -> D
T =  D -> E
```

Large Programs

```
L = W . Q
```

```
K = T . R
```

Architecture

```
M = L . K
```

The above description works at every layer.

SICP deals with abstraction and architecture at all layers of programming. It does not talk about real world idiosyncrasies of the system. For example it won't talk about whether the assembly instructions of the small program being executed is efficient and it won't talk about network latency.

dwaltrip 71 days ago [-]

> The design of small programs is isomorphic to the design of large programs at all levels

Good software design is not scale invariant. Significant scale deltas change things in fundamental ways. Designs that work and are maintainable, from a human perspective, at one scale often will fail at a larger scale.

Some designs scale better than others, this is true. But no design scales infinitely, and the vast majority of designs would start performing poorly at scales that are still relevant to human concerns

This shouldn't be surprising. Most things (nearly everything?) that we care about are not scale invariant.

AnimalMuppet 71 days ago [-]

I seem to recall something like the "rule of 10" - for every factor of 10 in scale, a new set of problems dominates. And, from experience, there's some truth to it.

crimsonalucard 70 days ago [-]

The concept of abstraction and process IS scale invariant.

What is not scale invariant is the real world details like I mentioned above.

1. What assembly language instructions a language compiles to

2. Network latency / topology / etc..

Scheme or other programming languages hides real world detail #1 so we don't have to deal with it (as much).

While there is a lot of work done for #2. we currently do not have as good abstractions for #2. as we do for #1. Therefore usually a programmer must exit the abstraction and deal with the real world issues. Hence the confusion. You usually don't have to deal with assembly language or even know about it because a good abstraction exists but you do have to deal with "network architecture" and know about it because no good abstraction exists (yet). This is what is actually going on. You are dealing with real world details because those details change. But the concepts of abstraction that SICP talks about remain invariant.

That being said my statement about SICP is correct. My diagram that illustrates abstraction and design from a high level point of view the way SICP does is correct. Even with real world details in mind there is nothing about my diagram that is incorrect. The diagram I drew IS scale invariant. SICP teaches such concepts and is thus in itself a good book that is also "scale invariant."

If you feel there is something about that diagram that is incorrect please point it out. If you can find a flaw without going into lower level details like network topology or assembly language then you can prove my statement definitely incorrect. But as such my statement is correct and the diagram is still correct.

The ideal is to come up with a language that can compile down into both applications and a network of computers with minimal leakage. The concept of abstraction that SICP introduces remains applicable to all layers even if we currently lack the technology to stay within the abstraction.

I think what's going on is you're just mistaken. You think that abstraction is not scale invariant simply because of the way we deal small programs and large programs is different. Actually it's the same, we just don't currently have the tech to hide the details for large architectures like we do for small architectures... work is being done on this front though... Terraform, docker, cloud computing all these things move it in the right direction but we don't yet have an abstraction over this area like we do for single applications.

This does not mean it's impossible. Who's to say I can't write an entire web app in one language and it compiles down to server app, networked physical servers, cache and database automatically? This is a very viable architecture. Why not let a compiler tune network topology like it tunes assembly instructions? Of course, it's not viable yet, but definitely in the realm of future possibility.

hliyan 71 days ago [-]

I agree. These theories should either be based on empirical evidence, simulations or built up from first principles in a rigorous manner. Otherwise they are about as concrete as philosophical treatises. Which is to say, not very concrete at all.

mirceal 71 days ago [-]

all these people have some experience in the field and they did put some things out that gave names/spread practices in our industry. is that enough to take everything they spew as the truth? far from it.

you should probably take a look and filter it through your experience.

notus 71 days ago [-]

They are speaking primarily from experience and I think they are pretty transparent about that. We're not in a field where many studies even exist on these types of processes. It is up to you as a developer to try them out and assess their merit.

tootie 71 days ago [-]

It's good point, but most of what I read on his blog feels like a distillation of best practices that I've seen mature in the industry. It all rings true and matches my experience of what works best. But can I prove it? No, not really.

hacker_9 71 days ago [-]

Agree. They talk the talk but don't walk the walk.

crimsonalucard 71 days ago [-]

I agree with you. There's another angle to look at it though other then "evidence based" or scientific.

In the field of engineering a large portion of it has to do testing to verify systems that are unpredictable or not modelled by theory. For example: building a plane involves testing it a wind tunnel.

The thing with computer systems is that computers are deterministic math machines. You don't need to test a computer system because it's output is 100% consistent and bounded by theory (most of the time). Thus you can use axioms, theorems and proofs to build your systems. This is not an "evidence based" or scientific approach to architecture this is within the realm of math and logic.

Unfortunately a large part of software is still an art and we still treat a huge portion of it as if it was a physical unpredictable system with unit testing. Theory is a bit underdeveloped in computer systems. Where it is underdeveloped is: any engineering problem that sports the term "design" is a problem space where no theoretical basis exists (or is known to exists by the user of the term). For example the "design" of "system architecture"

Amazingly such a theory sort of actually exists in Computer Architecture but is relatively unknown. It's newish, not concrete and it's not fully fleshed out in the context of computer system design. It's called Category Theory and it's mind bending because it turns this vague concept of "design", "architecture", "abstraction", "composition" and "process" into a language that is formal and well understood.

Though I don't think it's part of category theory yet, and I'm still a bit of beginner, but a formal language for system architecture at every layer allows for a different way of solving problems. Instead of "designing" solutions adhoc, we could be able to "derive" solutions. Given a problem you would be "deriving" a system architecture as a solution rather than designing one.

This is what I see missing from every single one of Martin Fowlers lectures. He treats this stuff like it's 100% art, and while it largely is, the overall goal of architecture should be an application of Theory, Science and Art.

Theory is priority, it says definitive things about a system, Science is secondary as it gives quantitative probabilities about things not described by theory, art is tertiary as whenever you employ art or your "design" chops you are admitting that you are operating in an area where no theory or science exists (yet). Martin Fowler treats architecture as if it was 100% in the domain of art, when the reality is it's actually the amalgamation of all three concepts.

sooheon 71 days ago [-]

> He treats this stuff like it's 100% art

Is this true? I'm just picturing in my mind juxtaposing Fowler's writings and I dunno, an art critic's.

Category theory is cool and important, but to insinuate that all design problems can be "derived" by it sounds like a mathematician declaring that the work of all civil engineers as just "applied math".

crimsonalucard 70 days ago [-]

>Is this true? I'm just picturing in my mind juxtaposing Fowler's writings and I dunno, an art critic's.

He does. The language is different. He uses technical buzz words from the tech industry rather then the jargon from the art industry.

A paper with science will have data and statistics. A paper with logic will have formal descriptions, axioms and theorems. Martin Fowlers papers contain little of any of these things.

>Category theory is cool and important, but to insinuate that all design problems can be "derived" by it sounds like a mathematician declaring that the work of all civil engineers as just "applied math".

I never said that all solutions can be derived from category theory, I meant that the current state of category theory hints at a possibility that this could be possible in the future.

Civil engineers do a lot of similar things to what a proper Software architect should do. They use science, theory and art to materialize their ideas. I'm saying category theory COULD be pointing at way of making that "art" section smaller or eliminating it all together. Though total elimination is IMO unlikely I think making it much smaller is very viable. This process is even more amenable to computing systems as the solution space is much smaller. For computing systems, I think it's more reasonable to say that it may be possible in the future for "art" to be completely eliminated from this field, still unlikely imo, but definitely more likely then other types of engineering.

sooheon 70 days ago [-]

> I never said that all solutions can be derived from category theory

But I think what you're saying is that they *should*. I don't understand how this is possible or desirable.

However formally described the internals of a system are, it must interact with the world, and designing those boundaries depends entirely on *what* you want your system to do, which is an art. Just like you can't get an ought from an is, you can't derive a solution to "will this satisfy the user" or "will this be done in time for market" from correctness proofs of code.

You might be referring purely to the computational soundness of a system, but that is to system design as the structural soundness of concrete is to the architecture of a bridge.

crimsonalucard 70 days ago [-]

>But I think what you're saying is that they should. I don't understand how this is possible or desirable.

Definitely not possible (yet) but this may be possible in the future. Whether it should be or not is not something I talked about. But since you brought it up, my opinion is that if there exists a function that can definitively derive the best possible system for a problem given a set of requirements then YES absolutely I would use it rather then design a system myself.

Whenever we design a system we have no idea whether that system is the "best" possible solution. Better to derive the best then to "design" or aka "estimate" a solution.

Anyway this stuff is mostly speculative not entirely worth debating over speculation. If it never happens what's the point?

>However formally described the internals of a system are, it must interact with the world, and designing those boundaries depends entirely on what you want your system to do, which is an art.

This is true. Designing the specifications of a system is largely an art because you could be trying to satisfy a customer who needs and wants are unknown and varying. This is not what I am referring too.

I am talking about implementing a solution to MEET those specifications. This part unfortunately is also largely an art. I'm speculating about a function that can potentially FIND the BEST solution to meet a specification and therefore eliminating the artistic portion of this part of the problem, not the specification part.

>You might be referring purely to the computational soundness of a system, but that is to system design as the structural soundness of concrete is to the architecture of a bridge.

The architecture of a bridge is different from software architecture. I am talking about the "architecture" of a system to meet the high level needs of a specification.

The product designer translates the world of vagueness and opinion into a set of exact specifications.

The software architect builds a system to meet those specification or potentially changing specification. I am referring to the process being implemented by the software guy, not the product designer.

The bridge architect usually is a hybrid role he does a bit of translating requirements and implementation as well. It's less clear cut.

To fit your analogy lets use a designer instead. The designer gives me a shape he wants the bridge to be in, and how many people are going to walk across the bridge per day. The civil engineer builds the bridge to meet that specification using the cheapest materials possible. The civil engineer is not concerned with whether or not the shape of the bridge is pleasing to the people who walk over the bridge. That part is the responsibility of the designer.

Think about it, even for systems implemented by the software architect today, even when the architect has knowledge about the EXACT specifications needed to be fulfilled by the system he is largely doing a lot of design work. He has no way of verifying whether his design is the best way to do things.

fogetti 71 days ago [-]

I don't really understand your comment. Shows your ignorance to be honest: have you ever heard of active records? Or optimistic locking? Because these are VERY tangible software constructs and Martin

Fowler for example wrote a whole book about them.

nafey 71 days ago [-]

The GP was about software architecture styles proposed/endorsed by Fowler et al and not their general contribution to software. Keeping this in mind its fair to ask for data/studies backing their claims.

afpx 71 days ago [-]

The evidence is out there. If you look at successful libraries and applications, they almost always gave well-defined design and architecture guidelines. In fact, I can’t think of a good software library out there that doesn’t have a solid design and architecture. On the other hand, I often see big piles of mud that have to be re-written over (and over), because people believe architecture takes too long or they believe it’s unnecessarily complex.

More