Hacker News new | threads | past | comments | ask | show | jobs | submit epirogov (16) | logout Ask HN: Software architects – what's your typical day look like? 198 points by jarl-ragnar 19 hours ago | hide | past | favorite | 95 comments A question for anyone out there working as a software architect (especially microservice architectures). What's your typical day comprise of? Where do you find yourself focusing your time? What resources have you found to be most useful?

azalex 17 hours ago | next [–]

The essence of my job as an architect is commonly described as a 'professional negotiator', implying that my primary responsibility is convincing people into doing the right thing (tm). My time is typically split between two main things: meetings and research/design work. On a typical day, I have 4-6 meetings with different groups of people. Some will be solution design discussions with engineering teams where we try to figure out how a particular challenge will be solved together. Others will be with product managers, talking about feasibility, cost and time estimation. Yet others will be with senior directors about long term strategy and infrastructure. Finally, a very important part of the meetings is mentoring. Knowing something is valuable, sharing that knowledge is invaluable.

While this may sound like a lot of meetings (and engineers typically abhor meetings) they are typically very useful and very rewarding.

The remaining time I typically spend on doing preliminary research, design, documentation and every now and then even coding, which I thoroughly enjoy.

reply

rcarmo 3 minutes ago | parent | next [–]

This. This is the "right" answer. reply

azangru 17 hours ago | parent | prev | next [–]

> convincing people into doing the right thing (tm). How do you deal with the burden of knowing/predicting what the right thing (tm) is / will turn out to be? I am constantly agonising over choices I'm about to make, and almost invariably come to regret some of the choices that I, or our team, have made (which include, but are not limited to, React, Gatsby, Flow, Stylus, Enzyme, Jest, SCSS, single-page app architecture, possibly graphql, etc. etc...). Not that I'm an architect; but still. How do you cope?

reply

azalex 16 hours ago | root | parent | next [–]

In my opinion, the fundamental difference between an architect and a staff/principal engineer is that architects are more generalists and principal engineers are more specialised. It does not mean, of course, that architects don't or can't have deep knowledge of specific areas or that principals can't have a wide understanding of their ecosystems, but an architect absolutely must understand the ecosystem they work in and a principal engineer absolutely must understand the specific technology stack of their choice. Understanding that every choice is a trade-off and comes at a price, that someone at some point will have to pay. You chose the latest and greatest web framework? Great! How many developers at your company will know how to use it? How long will it take for them to learn it? How big is the talent pool you can hire from? How much time will it take to migrate your products to it? Who will support it, is it "wild west" open-source or does it come with commercial support? How does it fit with your deployment model? Has it gone through rigorous threat modelling? There are about a billion factors that may affect your choices and finding the right solution will always be a central part of the problem.

To actually answer your question about how I deal with this, this is why I do research. Let's say our product people come up with a brilliant new product idea that we don't have an existing solution for, then I'll spend some time on finding information about the available options and take notes on the benefits and drawbacks. I will try to find people who are experts in the subject and get their opinions.

What I can not do, is to actually go and gain a deep understanding of every single language/platform/framework/library there is, I'm just a human after all.

reply

jvalencia 14 hours ago | root | parent | next [–]

It's probably a good thing if you're not too expert. If you were deep into it, you might be impressed by the choices, but miss the glaring red flags: e.g. The git repo for this cool API has no adoption, no stars, no commits in the last 6 months, etc. reply

atom_arranger 16 hours ago | root | parent | prev | next [–]

- Keep your code reasonably clean
- Use strong typing
- Make sure your tests don't test implementation details if possible
    - e.g. using more integration like tests written with react-testing-library
    - e.g. using visual regression tests
- If you sense something is off about the tech stack make a POC of partially migrating to something else as soon as possible
- If the thing you tried is better come up with a plan to migrate and do it as soon as possible
- If you've followed all the bullet points above then switching to a better solution shouldn't be that difficult That's how I try to deal with it. Also if you're not sure which thing is better do POCs with both solutions. Alternatively if you're working on a greenfield project and a week in things don't feel right don't be afraid to try rewriting it with one of the alternatives you were considering, it'll be a slight hit to productivity for a day or two but it'll pay off long term. By following these guidelines I've never felt on a project that I've painted myself into a corner with the tech stack, or chosen something very bad that is not reversible.

reply

mickeyp 17 hours ago | root | parent | prev | next [–]

That ecosystem moves very fast and... reinvents itself all the time. Here's another way of looking at that problem:

How many of those tools and technologies existed 1, 3, 5, 10 years ago?

Would you trust a six month-old tool on github to form the nucleus of your -- probably mundane, in the good way! -- project? Would you risk it all for the chance to have a slightly different way of rendering CSS or generating HTML?

... I didn't think so. React's been around for a long time now and it's got a lot of momentum and people who know it. So that's a pretty safe bet. Rinse and repeat.

Not all tools are critical and you do have some leeway. But when in doubt, pick something you can easily hire people to help you accomplish; that has been around a while; and that feels established.

reply

azangru 16 hours ago | root | parent | next [–]

> React's been around for a long time now and it's got a lot of momentum and people who know it. That was true for jQuery as well; and yet, it is clearly no longer the right thing (tm) to choose for a project. Not because it's no longer cool; but because the underlying technology — the web browser — has gotten so much better it is no longer relevant.

Same fate may await React. At least, with jQuery it's pretty straightforward to gradually wean oneself off of it. I don't know how firmly we are going to be stuck with React when its time comes.

reply

thrashh 13 hours ago | root | parent | next [–]

It wasn’t a change in web browser tech It was that someone put in the time to write a JS preprocessor that went along with their framework. That was React’s novel addition

Without a preprocessor, users of your framework either use plain JS to generate HTML (i.e. jQuery) or you use templates.

Both of them always looked really ugly on frontend

With React's JSX preprocessor, you can write HTML by… writing HTML. And you can mix custom components without doing some weird and ugly syntax

Does React use a shadow DOM? Yes, but so could have jQuery or Handlebars.js too, and it wouldn't have mattered. People don't care how it works as long as it works. As far as libraries go, what gets adopted is what looks and feels the nicest — as it should be

reply

azangru 9 hours ago | root | parent | next [–]

> Does React use a shadow DOM? Yes No. You must be confusing it with virtual DOM.

As for React, what we are currently observing is that 1) it's a fairly large library in itself (~50kB of minified gzipped javascript); 2) it has a relatively slow DOM update mechanism; 3) it rerenders too eagerly, making it too easy to further degrade performance; 4) it's a lock-in into a non-standard jsx syntax, forever tying you to preprocessors.

What conclusions an architect would make looking at all this, I do not know.

reply

Too 12 hours ago | root | parent | prev | next [–]

Jquery got squashed from two directions. One being react and other data binding frameworks, the other side being browser tech modernizing and aligning, such that bare js could be used, without a thick compatibility layer in between.
reply

bcrosby95 15 hours ago | root | parent | prev | next [–]

jQuery has existed for a long time, still gets updates, is still used in greenfield projects, and has a large body of expertise around it. The concern is abandoned projects that no one knows how to use, not that something is a bit old. React is beyond that point. It's not just the tech junkies hopping from the latest fad using it. There's been enough widespread adoption that there will still be React projects in 20 years.

reply

ARandomerDude 16 hours ago | root | parent | prev | next [–]

Nevertheless the point is the same. Some technologies are more stable and broadly adopted than others. A slice of the total risk management pie entails selecting a tech stack that is less likely to die soon than its competitors. reply

jupp0r 17 hours ago | root | parent | prev | next [–]

A great mentor once told me about the poker analogy method of coping with situations you describe. Let's say you are a great poker player. When you get dealt a good hand, you play it and end up loosing - does that make you a worse player?

Turns out even the best of all players loose quite frequently. What distinguishes them from others is that they make the best choices with the limited information they have at the time. They later go back and learn from any mistakes they have made and move forward.

reply

anon7725 14 hours ago | root | parent | next [–]

I ran into an interesting point about poker in a recent episode of EconTalk[0]. Really good players play slightly more hands than they should (ie on a tournament table with 9 seats, a good player might play 15-20% of hands). Amateurs play way more hands than they should - like 50% or so. Translating to this thread, I'd map "playing a hand" to committing real resources to a project (switching tech stacks, etc). Pros will be somewhat conservative and only take that bet when they have a higher expectation of success. [0] https://www.econtalk.org/annie-duke-on-the-power-of-quitting...

reply

ElevenLathe 10 hours ago | root | parent | next [–]

In other words, you gotta know when to hold 'em, know when to fold 'em, know when to walk away, and when to run.
reply

comprev 16 hours ago | root | parent | prev | next [–]

Similar to management roles in the armed forces - those on the ground have to make split second decisions based on the latest information they have to hand. Getting it wrong usually has fatal consequences and it takes a certain personality trait to continue making those decisions. reply

e12e 14 hours ago | root | parent | prev | next [–]

Many good responses here already - but I'd say that eg choosing between react and vue or svelte isn't an architectural decision. But choosing between server rendered php and react is (is this a series of web "pages" with some forms, or is it an "app"?). Choosing between mysql, sqlite and postgresql may be (do we need row level security? Would mysql multimaster benefit us? Do we expect to deploy many, small instances where the simplicity of sqlite will shine) etc.

There are many decisions to make building a solution - the architecture part has to do with fulfilling the requirements.

The tech-lead/senior developer/team lead part has to do with implenting the architecture - our team knows vue, so we go with that to implement the app. We're invested in ms sql so we prefer that to postgres - but sqlite would be a poor fit.

reply

gautamdivgi 12 hours ago | root | parent | next [–]

My personal take is I would go a level higher and define data consistency and availability requirements. Provide them to dev teams and if the teams need help meeting them I provide recommendations. I generally try to keep design out of architecture. It’s what you need to satisfy product or service needs. The technology gets into how you want to implement. I try to leave that to the tech lead for the team.

reply

jononor 10 hours ago | root | parent | prev | next [–]

Realize that a perfect solution is not feasible (it does not exist). Not even an optimal solution for your particular problem is feasible (too many unknowns about both the problem and potential solutions, impossible to uncover and analyze them before hand). Realize that if you had chosen another solution, you would have other issues. Not "no issues". Probably not even "fewer issues" or "less severe" issues - just as high a chance

Realize that most technology choices actually have very little impact on the business. If one uses A or B, it does not matter much to do whatever it is your company does. Very few tech choices have the potential to kill the company/unit/product.

Realize that once a choice has been made, worrying about it does not help anyone.

So the goals are not perfection. Not anywhere close. Just something that:

1. lets us move forward now (avoid decision paralysis)
2. not worse than before (if so, just stick with what you got)
3. good *enough* to deliver the needed business value
4. does not paint ourselves into a corner. Then we can always fix and improve things later

reply

0xbadcafebee 11 hours ago | root | parent | prev | next [–]

> I am constantly agonising over choices I'm about to make You cannot predict the future. It is impossible know which choice will result in better outcomes. Whatever you choose has probably about equal chance of success or failure. So just make a choice, with a general idea of what you will do when the choice fails. Most choices are not make-or-break, so you should really not be agonizing over most choices.

reply

ycombobreaker 6 hours ago | root | parent | next [–]

> It is impossible know which choice will result in better outcomes. Whatever you choose has probably about equal chance of success or failure. It's a common fallacy to assign 50/50 probability to unknown distributions. Furthermore, many choices are not binary, or may not be as simple as right/wrong in hindsight.

But I agree with the idea of amortizing your perspective over many choices.

reply

gardenhedge 17 hours ago | parent | prev | next [–]

How do you personally deal with designing solutions where you haven't been able to convince people to do the right thing? For example where there are limitations imposed on you or someone higher up makes a call that doesn't sit well with you reply

azalex 15 hours ago | root | parent | next [–]

It does happen, quite often, to be honest. The single most important thing in this case is to understand why it happened. A typical example is when we come up with a design proposal that I think is a good one, but turns out to be that the estimated cost of building and running it outweighs the projected revenue the product will generate. Another example would be when I want to have a particular design implemented by an engineering team but they don't have enough capacity (people, time, etc) to deliver it the way it was intended. There have been cases when it was all about politics (sadly, I must add, but it does happen), i.e. someone in higher places of power want to push their own agenda and you get overruled. These are cases when the previously mentioned 'professional negotiator' role gets really important. These are the cases when you need to go to the people involved and try to understand their priorities and their constraints. Once you have the information necessary, most of the times you'll be able to come up with an alternative solution that is acceptable for everyone. Sometimes you can convince others to give way, sometimes you'll need to.

And then there are times when you just can't do anything because, for example, you get overruled by an overzealous CTO. In these cases, as sho_hn also pointed out, the best you can do is to make them understand the implications and brace yourself for damage control, because the fallout is going to be on you anyway :)

reply

sho_hn 16 hours ago | root | parent | prev | next [–]

Make sure you negotiate the responsibilities of your role so that you can say "I'm not able to sign off on this". You may get overridden anyway, but the consequences must be known. And if you got people to agree that you should be able to say this, they usually understand the gravity of you having to invoke it and will live by it. reply

stronglikedan 12 hours ago | root | parent | prev | next [–]

CYA. Send an email to the higher up stating the negative impact that their decision will impose, and ask them to confirm that they would still like to proceed. Then don't do anything related to that decision until they respond. reply

weard_beard 9 hours ago | parent | prev | next [–]

Same. This is a great depiction of a healthy practice. reply

angarg12 14 hours ago | prev | next [–]

A little offtopic, but some people might find it interesting. My company doesn't have a "Software Architect" job title. Instead the closest thing is "Staff+ Engineer", which does pretty much the same things. However there are some key differences.

First, these aren't "Ivory tower" architects. They still write code (although in practice not much) and stay connected to implementation details, mostly via code and design reviews.

Second, architecture is every dev's responsibility, at different levels. So Staff engineers might oversee the architecture of entire systems spanning a whole org, while senior engineers oversee a single team. A mid level dev might design the "architecture" of a small service integrated in a larger system.

Staff+ engineers usually report to senior leaders and aren't part of a team. They might hold office hours and break disagreements within or between teams.

I'm not one myself but in a conversation a Staff eng. told me this was the first job where no one tells him what to do. Instead he spends his time proactively looking to solve business problem. This might be top down (e.g. take a business problem and try to find how to solve it) or bottom up (e.g. take some new tech or tool and figure out how it can be used to achieve some business goals). Often problems can also be solved without tech at all (e.g. aligning stakeholders).

Lastly at this level engineers are expected to be leaders as well. Mentoring, sponsoring, etc. is pretty much a requirement. They should be force multipliers, making other engineers around them better. They might also scale themselves by producing content, such as tutorials, talks, training, etc.

reply

DoingIsLearning 11 hours ago | parent | next [–]

> My company doesn't have a "Software Architect" job title. Instead the closest thing is "Staff+ Engineer" Not off-topic at all, this is aligned with what I see. In older or 'regulated industry' companies the role and career path of Architect still exists. But more and more I just see senior roles that are flavours of Principal/Staff/Fellow and they still have normal Engineering duties but then in early greenfield stages of projects act as Architects, just not as a full time role.

reply

jslakro 8 hours ago | root | parent | next [–]

Training sphere also reflects this trend and nowadays is less common to find content focused on "Software Architecture" roles. It seems more like a thing from the past. Agile, horizontal teams and even distributed systems are motivating that specific role to disapear reply

danielmarkbruce 10 hours ago | parent | prev | next [–]

Is there any modern (let's define it as "started in the past 10-15 years") product company that has this "architect" role? I was under the impression it's antiquated. reply

lexx 18 hours ago | prev | next [–]

- Always on a state of mind of redesigning and simplifying everything
- Spending a lot of time trying to figure out the business better
- Studying a lot of books, articles, codebases and discussions on software architecture
- Inspire the team and younger devs to avoid complex tools and solutions and stick to the basics
- Balancing everyday and urgent business needs while leading towards a more simple, boring and maintainable system.
- Trying to persuade people to avoid technical debt at all costs

reply

dimgl 16 hours ago | parent | next [–]

- Inspire the team and younger devs to avoid complex tools and solutions and stick to the basics
- Balancing everyday and urgent business needs while leading towards a more simple, boring and maintainable system.

- Trying to persuade people to avoid technical debt at all costs

This is literally my job everyday now as a principal. I enjoy it, but it's quite grueling, tbh.

reply

discussDev 14 hours ago | parent | prev | next [–]

This is what an architect should be doing. Too often people are made architects that never had to maintain a system more then a few years if that and aligning new tech for every problem until you end up with an unworkable, inconsistant, mess that can't be easily estimated. reply

throw1234651234 17 hours ago | parent | prev | next [–]

"Studying a lot of books, articles, codebases and discussions on software architecture" - like what? 90% of it is "write clean code", "put in a cache", "normalize your db", "use a queue if the service is really spotty/unreliable". "Fundamentals of Software Architecture" is 95% fluff. So is "Clean Architecture". Most patterns like CQRS, DDD, fully event-driven etc are complete overkill and a sign of a BAD architect unless there is a real cause for it. 90% of the job is NOT overcomplicating.

"Trying to persuade people to avoid technical debt at all costs" - put in a linter and concrete rules (even if a rule is loose, note that in writing) or devs will get upset and think you are singling them out.

reply

nrawe 13 hours ago | root | parent | next [–]

Those approaches are not “fluff” or overly complicated if the problem-space warrants them. There are of course limits. If you want to write a script for use once or twice, worrying about coupling and cohesion is probably not worth your time. If you are building a system to last 2-5 years out, it absolutely is worthwhile.

If you have a dead simple CRUD problem with very little business rules/operations then DDD isn’t right either. But if you have a complex set of interactions, state, and policies it’s the right call. (The big red book of ddd has a great table on exactly this)

But there are problems out there that are suitable to those approaches. The devil is in the detail; unfortunately a lot of strategic design goes by the wayside to cargo-cult zeal.

But the same can be said for the other way of thinking. I’ve worked/work with engineers who think any form of abstraction is abhorrent and that they don’t need to care about quality because they’ll delete it all and start again (they’ve not tended to care about the opportunity cost of that to their employer, either). Any database interaction looks like ActiveRecord and all technology choices are immutable.

The key in all things is balance. What is simple vs overwrought is almost entirely down to what the framing of the problem is.

reply

lexx 17 hours ago | root | parent | prev | next [–]

Also, I agree with you about over-complicating with all those patterns. But before I am able to dismiss a solution, I have to understand it. Trying to keep it simple is hard reply

sidmitra 17 hours ago | root | parent | next [–]

@lexx, i think most of the things you put in your answer were very reasonable and non-controversial and probably what most good senior-ish engineers do. I think the parent comment is slightly in his "local minima" and he's generalizing his experience to everyone else

> "Trying to persuade people to avoid technical debt at all costs" - put in a linter

This for example does not make sense to me. This would mean all tech debt is just static analysis. There is no linter for figuring out the correct abstraction and using them correctly. Eg. no linter yet will tell you "hey you should have used a state machine here!"

You need a width range of experience before you can develop "better" taste in what's good and bad in different scenarios. Some of it can be gained from years of experience, and some from exposure to books/code.

reply

throw1234651234 17 hours ago | root | parent | next [–]

Definitely nothing against what Lexx said. I did want to call out that most architects over-complicate in my experience. What we do just isn't as profound as people make it out to be unless you are genuinely working on some latest ML for billions of revenue or stabilizing SpaceX's landing thrusters, sequencing DNA, etc. reply

SgtBastard 11 hours ago | root | parent | next [–]

Having a broad and deep understanding of a range of design patterns in an otherwise banal enterprise environment is critical when the existing technologies you work with are numerous, decades old and have built up like layers of sedimentary rock.

> in my experience

Friend, you're making it clear that you don't have much.

reply

doctor_eval 12 hours ago | root | parent | prev | next [–]

> 90% of the job is NOT overcomplicating. I totally agree with this, but
>
> Most patterns like CQRS, DDD, fully event-driven etc are complete overkill and a sign of a BAD architect unless there is a real cause for it.

Once your application gets to a certain size, DDD, CQRS, and events will make your life far simpler.

Avoiding these patterns is a bit like saying that the solution to moving a piano is to get a bigger bicycle.

reply

lexx 17 hours ago | root | parent | prev | next [–]

Apart from those best seller books, the are great podcasts and articles that people discuss their experiences in depth. It helps me a lot, to compose a more holistic thinking. Technical debt does not increase only for technological reasons but also from business expectations. You can't just "build" a new garden and leave it be. You have to maintain it also. Keeping the system clean and well oiled before building more is important. But these concepts need to be reminded everyday to survive

reply

nateee 16 hours ago | root | parent | next [–]

Can you list some recommendations? For podcasts and some articles/authors to follow. reply

lexx 14 hours ago | root | parent | next [–]

Here are some of my current material:

- I have a great time listening to "Software Unscripted" podcast.
- On youtube the channel "code opinion". Although I don't like or use DDD and C#, I think that it has high quality and rewarding content.
- I am currently finishing "A Philosophy of Software Design" by John Ousterhout.
- Regarding articles, I read stuff related to my day's obsession or problem. I can't point you out to one. I mean, what do you need to evaluate/learn?

If you need something more specific topic-wise I am glad to share with you

reply

moris_borris 17 hours ago | root | parent | prev | next [–]

> "Trying to persuade people to avoid technical debt at all costs" - put in a linter and concrete rules (even if a rule is loose, note that in writing) or devs will get upset and think you are singling them out. I went from a company with no linter or style guidelines and PRs were an absolute nightmare. Eight devs, eight completely different styles, each insisting theirs was right. I wasn't able to turn stones into loaves (i.e. I wasn't a Senior Dev at the time), so my style was categorically incorrect. I switched to a company with an aggressive linter, and all that superfluous debate over tiny patterns and "personal preference" went out the window. I no longer dreaded PRs.

reply

throw1234651234 14 hours ago | root | parent | next [–]

I think that's the main benefit by far - avoid useless debate. The icing on the cake is that I can write code in my style (tabs instead of double spaces, parens on new line, whatever) and then have my IDE or a CLI command refactor to the team style before putting up a PR, so it's really not an issue. I usually default to some large company's style guide so it's difficult to argue against. For example, for .NET we use Microsoft's style guide for React we use Facebook's, etc.

reply

mickeyp 17 hours ago | prev | next [–]

My view as a "full stack" architect and developer. Depends on the size of the firm. Very, very large ones will see you write little code -- generally to the detriment of the company, the architect, and the teams they interact with.

Good architects are hands-on. They write proofs of concept implementations before recommending a tool or technology.

Bad architects dream up nothing but gormless and inscrutable charts and diagrams that, like crap ink on vellum, slowly fade into illegibility in a remote Sharepoint server.

Good architects clarify and explain. Their job is to assist teams in melding disparate technical viewpoints with that of the business -- who themselves often lack focus. A good architect facilitates that: they'll rarely look at lines of code, per se, but more the general trend and direction.

Bad architects dream up complications that serve no one but their own ego. I once worked with an "architect" who spent most of his time mapping out some data, represented in JSON, using eBNF -- a tool used for specifying formal grammar for computer languages. It could've been solved with an example + some data typing in the margins. Don't be like that dude; people struggled to make sense of the trivially simple data we had to store. For larger things use a schema-like structure; for smaller things, trust that people and teams are not stupid.

Good architects are skilled at everything. They've hacked up CI systems in batch or bash files, before the current tooling even existed. They've built all manner of systems, in a range of environments and possibly languages. They're pragmatic,

and can spot trouble a mile ahead.

Bad architects envelop themselves in what ever the buzz du jour is: like microservices, which is never an opening gambit for 99.9% of problems you're going to encounter in a greenfield project. They're the ace up the sleeve when all other performance and scalability efforts have failed. Good architects know when to tell a tech lead to lay off the technobabble: chroming your CV is not in the interest of the business as a whole.

Good architects prefer plain, old working tools. That usually involves a RDBMS. Sometimes many; and often times you need to make existing ones talk to other things. Bad architects thing old means bad. XML is not bad; what people did with it was, for example.

reply

P5fRxh5kUvp2th 15 hours ago | parent | next [–]

hard agree with everything you've said here. At my current company it's actually a fight as it's not necessarily the architects themselves that want to shit out diagrams all day.

It's a part of the job, for sure, I'm just pointing out it's not always the architect.

but your points about hands-on, respect simple, and having a wide base of experience is 100% spot-on imo. I do agree with your point about the diagrams, just wanted to point out that's sometimes due to pressures rather than wants.

reply

mickeyp 14 hours ago | root | parent | next [–]

Agreed with the diagrams. But, if your job is to convey information, and non-techies prefer that method, then... well, you abide. And charts *are* useful for sure. reply

ftkftk 10 hours ago | root | parent | next [–]

In my experience a few boxes and sticks usually communicate more effectively to more constituents (both technical and not) than the most well-written document. The trick is to know how much detail is necessary to include at the time. reply

dr_kiszonka 5 hours ago | root | parent | next [–]

Not having this skill is the bane of my existence! There is always a person who complains* that I provide too little detail ("you need to be more rigorous/specific/careful how you define") or too much of it ("we are getting too much into the weeds here" or they just tune out). I asked a much more senior person for feedback and it is always too much or too little when go from one revision to another.

- Yes, it is better that they articulate what they don't like than just completely ignore it and leave me in the dark.

reply

f1shy 17 hours ago | prev | next [–]

In my current employer "architect" are people that do (exclusively) drawing. The preferred tool is IBM Rhapsody. Once somebody suggested to use PlantUML, he was directly laugh at his face; and they explained how much better Rhapsody is. To be fair, they had some good points... But once I suggested they should do some back and forth with the people actually writing code... they told me "I cannot understand code, and it is all right so. I do not need to. I'm the architect, not the carpenter"

Another time I asked the chief architect --we were working in an embeded system-- "I have to program this function, what is my RAM, ROM and CPU budget for it?" His reply shocked me: "I'm architect here, I do not care about that little things, I do only the architecture, look here" and he proceeded to show me lots of boxes connected with lines. Then he added "that here is what is important, the real architecture is here".

Needless to say, those projects failed miserably, with no hope for recovery...

reply

reacharavindh 17 hours ago | prev | next [–]

I’m not an architect, but used to closely work with one and it was a pleasure. The key role that our architect played that brought immense value to us was being the integrator of teams. Teams had a high level of autonomy when it comes to implementation details which meant there could easily be “reinventions”, incompatibilities, and even narrow minded choices. Our Architect always had the big picture view across teams, almost always sat in our feature grooming sessions with valuable inputs, worked closely with product owners who bring in feature requests that may need integration of multiple services etc.

Above all, he was very active, and approachable to all teams instead of hiding behind barriers and occasionally bringing out UML diagrams. Several times, when we were debating which technical choice to make, held chime in with valuable points about what would be better if we were to integrate with another service, or something like “that team did this because blah, which might be relevant to you”.

reply

FlacoJones 18 hours ago | prev | next [–]

- It's typically a journey up and down the stack in terms of frontend, backend, and DevOps, and a journey in and out of solving inherently complex problems (where the seams of your services are, authentication) and incidentally complex problems (frameworks, machine-specific issues, Docker hell etc.)

- 80% what needs to be done today/what fires need putting out, and 20% planning for the future and seeing how the bleeding edge can legitimately help us (e.g. WASM coming soon...). This is part of the general keeping up with advances in industry

- Mentoring, jumping on pair programming calls to unblock other teammates.

reply

croo 18 hours ago | parent | next [–]

This reflects my experience as well. Roadmap plannings and API designing between systems, communicating and enforcing best practices within the time constraints, meetings with different service developers/designers to solve the upcoming features... Less coding, more meeting and reviewing, overall clear view of the big picture is a must.

reply

mrweasel 18 hours ago | parent | prev | next [–]

It sounds like you're very good at what you do. Sadly I've seen "architects" being more focused on drawings, crude prototypes and fluffy ideas. Basically dreaming up ideas and concepts, throwing them over the wall to developers and SREs to pick up. A previous boss of mine had to callback a candidate for a job, telling him, that despite us hiring pretty much anyone, we had no need for someone who thought of software architecture as a desk exercise.

reply

ryanlitalien 13 hours ago | root | parent | next [–]

These fluff and idea types are typically called "Ivory Tower" architects (from my experience). reply

rlyshw 18 hours ago | parent | prev | next [–]

Seems like a lot of ongoing o&m and sustainment of an existing product, similar to any other day-to-day. Do you have insight on what the proposal, design review, and first commits look like when architecting a brand new project? Like what do the first 30-60-90 days look like from product idea up to those o&m and sustainment activities?

reply

diarrhea 14 hours ago | parent | prev | next [–]

What's Docker hell? reply

daigoba66 18 hours ago | parent | prev | next [–]

This is basically my own experience as well. reply

Rabidgremlin 10 hours ago | prev | next [–]

I've had an "architect" job in some form or another for 20+ years, so have learnt plenty of things that would have greatly surprised my younger self... My day job is basically to be a "force multiplier" and it boils down to:

- cat herding = meetings, discussions, negotiation, "shoulder to cry on" = way more soft skills then I ever thought I'd need
- Big picture stuff = "town planning" for tech, second order thinking, pulling together cohesive plans/strategies, principles, constraints = way harder to effectively communicate this stuff than my younger stuff would have thought
- rapid altitude changes = dropping from the 10000ft view down to helping a team troubleshoot some production issue, helping a junior dev with a code issue, solving a dispute between devs, hands on evaluation of some new tech, then jumping back up to talk to a leadership team about some new grand strategy, or to planning out a multi year program of work = a ton of skills that my younger self would have never have guessed at, finance, budgets, "business" language, operating models along with keeping my technical skills sharp

In terms of resources:

- Anything around learning to story tell and communicate effectively
- The "97 Things Every Software Architect Should Know" essays
- Your tech skills - write PoC apps, side-projects, try out new tech, learn to quickly grok strengths & weaknesses of tech

reply

marifjeren 17 hours ago | prev | next [–]

Noticing a theme already.. architects are fun killers! "Good architects prefer plain, old working tools." https://news.ycombinator.com/item?id=33880398

"Fight the urge to use 'latest and coolest' technology. (…) Stick to what works. Even when it's boring." https://news.ycombinator.com/item?id=33880978

"Inspire the team and younger devs to avoid complex tools and solutions and stick to the basics" https://news.ycombinator.com/item?id=33880224

reply

zerkten 16 hours ago | parent | next [–]

> architects are fun killers! I strongly disagree with this. Fun killers are working late nights to remedy poor planning, or getting paged multiple times every night because people didn't make informed choices. It kills the fun for the employees as well as others they interact with.

A good architect includes someone who understands the needs of the people in terms of growth and satisfaction. They'll push back on schemes that introduce toil and try to align around good bets that contribute to your skill set. That may involve blocking use of technology X which has Y long term consequences. The consequences of that long-term choice need to have counters to keep the people engaged, so the good architect considers how to manage that. They don't always get that right and we have some less well-rounded architects, but we should aspire to doing this.

reply

gardenhedge 17 hours ago | parent | prev | next [–]

Most likely because they are:

- Experienced devs who have "seen it all"
- Have to deal with the consequences and have more responsibility

reply

rr888 16 hours ago | parent | prev | next [–]

That is what it should be. From what I've seen most architects aren't like this, they play with whatever is cool then move on to the next company while the rest of us try to untangle the mess. reply

djmips 15 hours ago | root | parent | next [–]

Let's say 'Good architects are fun killers' - but really there is still a lot of fun to be had - and if not maybe one should move to a more mentally challenging job haha. reply

matus_congrady 17 hours ago | prev | next [–]

I believe that the job description differs a lot from company to company. From my experience, the best architects spent most of their time focusing on these things:

1. Always understand what the current business priorities are.
2. Design the overall system architecture so it covers all of the business needs, and nothing more. Always focus on keeping it as simple as possible. Even when people argue "lets do this some other way, because we might need it in the future".
3. Fight the urge to use "latest and coolest" technology. Don't listen to consultants selling you unnecessary tech. Stick to what works. Even when it's boring. Explain it properly to your team.
4. Maintain proper documentation. Focus on making it as simple as possible, easy to adjust and up to date. Otherwise its value will be close to 0.
5. Use UML diagrams. Think about where the complexity of the given system is, and use only the most relevant ones. For example, if the complexity lies in complex workflows, use sequence diagrams or state diagrams. If the complexity lies in complex datamodel, use ERD diagrams. But more often than not, architects also have to focus on other things, mostly related to DevOps, Platform Engineering and overall developer workflows. [shameless plug] That's one of the reasons I founded https://stacktape.com - AWS-focused "internal developer platform" that allows developers to do this sort of stuff on their own, without involvment of architects, DevOps or others.

reply

pelagicAustral 17 hours ago | prev | next [–]

Well I work for a small government and my responsibilities tend to be a little less structured or more fluid in nature. I have to adapt at times and crunch code just as much as I scope requirements and write technical documents. Meetings here and there, lot's of talking to people and lots of reading. Currently I'm working on a digital transformation strategy and I can definitely see myself writing a whole lot in the next few months to gather traction on the implementations that are needed.

I would say that at least for me, becoming a Software Architect has meant that I need to split my time between groundwork and strategising with an added layer of complexity that stems from the fact that I work for government, so there are a number of policies and internal bureaucratic barriers to get used to.

reply

jslakro 15 hours ago | prev | next [–]

Other related and recent question with useful comments https://news.ycombinator.com/item?id=33713075 reply

juancn 14 hours ago | prev | next [–]

I'm an influencer. I have no formal power in the sense that people are not supposed to do what I say, so I have to be convincing, and influence others behaviors to make the software better.

This implies analysis and compromise, having meetings, mentoring engineers, and quite a bit of coding and research.

My work is less guided that when I was a junior engineer, in the sense that I'm meant to discover opportunities for improvement and give guidelines. This leaves time to pick what I should be doing next, where I'm adding the most value.

This changes from time to time, it may be figuring out a new architecture, fixing a bug, getting involved in product development or handling an incident.

An important part of the job is getting involved in production incidents, this helps grasp where are we struggling and opens the opportunity to guide development to a state where customers are happier and operational costs are lower.

You also start to be more conscious about company finances (how much it costs to keep the lights running) and start thinking in ways to get more from the same resources.

Even if I have no formal power, people tend to pay attention to what I say and suggest, so I need to be careful, because an off-hand comment can have a lot of impact.

Architectural roles live and die by their word. We lead by influence, so be cognizant of perception and careful when you communicate with others in every interaction.

reply

ryandrake 13 hours ago | parent | next [–]

I think "influencing/leading people who do not report to you" is one of the hardest, most underrated part of mid-level people's jobs. When you're a low-level Individual Contributor, it's easy: You're not expected to lead anything or influence people. Take your ticket, do a great job at it, then take the next ticket. You're given a task, design it, and implement it, and job well done. On the opposite end, if you're a director or senior manager with a big org of people who report to you, it's also easy (or seems easy from my vantage point): You simply say "we must do this thing" and people do it. In the middle, we don't have direct reports, and nobody has to listen to us, yet our job is to convince people to do the right thing. Very difficult. You need to be political, cognizant of perception as you say, you need to gather favors and then call them in, prioritize and horse trade for things that are important, and so on. Difficult job. reply

gorgoiler 11 hours ago | prev | next [–]

First up: having total visibility of the org is really important. We have a hundred engineers and I see everyone’s changes at the point they make them public (in whatever form they are called today: pull requests, merge requests, diffs, eh.) Unless I engage then that’s the first and last time I see their stuff but it’s enough to stay on top of everything everyone is up to. It’s a surprisingly low volume / high value channel of information. Leave the code review to the experts in each codebase — most changes will be on point and need only a small or large amount of alignment before they can land. (By contrast, very few things need to have the brakes put on them.)

Some changes though will correlate with other problems across the codebase and this is where you should be stepping in to spot future patterns or current anti patterns and providing solutions and directions forward, or at worst, road blocks.

Once you have enough of these under your belt you can start proactively spotting hot spots in the eng org that need focused effort. More mypy typing for a core library. Two libraries that should be one. One library that should be two. Vertical slices of functionality in two products that should be horizontal slices pulled into a separate service (or, my preference, library.)

Processes are important too. You’ll see what people are repeating and or finding hard, and which could benefit from some love. Build infra. A testbed for debugging a process that is otherwise too heavyweight with production data. Teams that don’t talk enough pre-PR. Managers that need help managing consistent poor performers.

I think many people wanting to go into an architect career think it is highbrow design work. In reality, I’ve found you end up doing much more support / boiler room / janitorial work. It’s very satisfying, and very much reminds me of my career sidetrack as a school teacher. Above all, you are there to help people and teams reach their potential and get the most from their jobs.

reply

awesomegoat_com 18 hours ago | prev | next [–]

I am architect on the sales side. That's little bit crazier, I would guess. In the morning I look over our github/mail/slack queue to see whether someone needs urgent help.

Then, if there are customer meetings scheduled on that day, I frenetically try to re-create customer's infra in our lab (be it Anthos Cluster, AWS Systems Manager, Azure Arc, or istio). If there are no customer meetings I try to improve our stuff on github. Sometimes, I publish new blueprint, kubernetes operator, or a terraform module, or try to create new CFT resource.

reply

vbezhenar 16 hours ago | prev | next [–]

I'm not sure that I'd qualify as a software architect. I work in a small software company and basically I do all important IT decisions. I'm kind of architect and full stack developer and devops at the same time. So sometimes we do some design meetings. I suggest how to better design database, etc. I'm trying not to dictate everything but rather catch mistakes which are obvious from my experience. Avoiding mistake early is very important IMO.

Recent months I designed Kubernetes cluster. We have plenty of small and medium services and environments thrown around in a few dedicated servers. It's a giant mess. I dread when I need to untangle it. So it was unavoidable to redesign devops from the scratch and I decided to go with Kubernetes. I basically tried few approaches, some turned out to be too complex, e.g. I built a complete automation on terraform, shell scripts and flux and ditched it out, because nobody but me would understand it and I don't need that kind of job security. I ended up with a simplest setup possible. Terraform provisions servers (we use hoster with OpenStack), then I manually run kubeadm with prepared configs, then I run few shell scripts to install important stuff with helm, then I install our services with kustomize. I think it was very nice outcome. Simple and approachable for everyone who knows basics of Kubernetes. Not full-fledged GitOps, but I decided that we're not ready for it yet. That was one example of project that I did myself because we didn't have necessary expertise.

Right now I'm adapting our projects to work with Kubernetes, improving builds, adding health checks, writing yams, etc.

I also made a foundation for some important core projects which I couldn't trust others to design. Then people continued my work. It seems to work fine so far, I oversee those projects to keep them in a good condition.

Also I chat a lot with developers who stumble upon hard issues and struggle to resolve those or they can't make some decision.

And I have some vision how our system as a whole should look like in an ideal world. We don't have resource to implement that vision and probably never will, but I consider it a good direction so I'm trying to point important decisions to that direction.

Also I sometimes walk over repositories and fix stuff I don't like. Usually devops stuff, like bad dockerfiles, missing dockerignores, outdated dependencies.

Sometimes I feel like a janitor, LoL.

reply

doctor_eval 12 hours ago | parent | next [–]

The thing is that in computing, the “janitor” stuff is the hardest, and needs to be done by the best people. Keeping things clean is really tough, and only people who understand what “clean and simple” means, can do it. Outside of actual system design and writing code, I have spent a huge amount of my time over the last few years doing things like setting up CI/CD, E2E testing, writing templates, etc. Because it’s really hard to get this right, to have a vision for how things should be.

My point is that it sounds like you’re doing an awesome job. I’m sure you qualify as a software architect.

reply

ilikerashers 16 hours ago | prev | next [–]

Usually have differing work depending on which phase of project lifecycle. Early stage days are estimating + research. Often sitting with users and figuring out system complexity (from a usage and implementation perspective). Most important part of any project/transformation. Can never know enough. This is usually 2-3 hours meetings per day with users/technical groups, 3-4 hours documentation pieces.

Later stage projects are process heavy. 3-4 hours with support, incident managers, networking, testers, devs, SLA stuff, NFRs all that fun stuff. Rest of the day writing.

Overall it can be busy with long presentation prep and discovery/planning work or it can be quiet (something gets delayed) where I just go play with some new tech for a few days!

reply

chasd00 14 hours ago | prev | next [–]

A lot of meetings across many teams and a lot of powerpoint for those meetings. It's not unusual for an entire week to be completely booked back to back with meetings. The upside is I get to meet and work with lots of different people regularly so I learn a ton as well as contribute. The downside is lots of powerpoint. I'm also the last point of escalation for a handful of dev teams. That's the fun part, getting to work on problems, mentor, and help out. I really enjoy mentoring and watching inexperienced people blossom and start mentoring others and passing on what they've learned. It's very fulfilling.

reply

FourthProtocol 16 hours ago | prev | next [–]

I've been doing architecture since 2003 and I've written down the things I do regularly as an architect - https://www.wittenburg.co.uk/work

reply

stcroixx 13 hours ago | prev | next [–]

Doing non architecture stuff. Reviewing designs, troubleshooting issues nobody else could solve, mentoring juniors. I wouldn’t say there are any particular resources I consult when thinking about architecture, but being well versed in design patterns and knowing how to make standard diagrams is key - it’s all about communicating ideas and outside text, those are the tools. reply

gryf 13 hours ago | prev | next [–]

Sitting on Zoom with a permanent facepalm wondering how the outsourcers delivered a monkey when we carefully specified a lion. reply

comprev 16 hours ago | prev | next [–]

There is much diplomacy involved as the wider the scope of the project, more teams are involved, resulting in even more "right" ways to do X. Although I'm not an "architect" by title, things I build do impact many teams as we move towards standardising how to do X.

reply

onion2k 18 hours ago | prev | next [–]

I'm not an architect at the moment but when I have been I spent most of my time negotiating with stakeholders to work out what they really need rather than just planning what ridiculous nonsense they believed they wanted. reply

bravetraveler 16 hours ago | prev | next [–]

I'm responsible for both hardware and software to a degree Most of my time is meeting. Either other teams or my own.

Between these, I'm either building proofs or managing services we never found owners for with a reorg

reply

WhitneyLand 17 hours ago | prev | next [–]

Maintaining time for a good amount of coding. Good architects write code or become more disconnected and more similar to management.

reply

motohagiography 17 hours ago | prev | next [–]

When I was an architect, my job was to scale my knowledge and save others time. Not 10 minutes of googling, but several weeks or more for a team who would need to sound out a solution. An example would be spending an afternoon designing a slide that represented the main transactions we were building for as an M/M/1 queue (or other queue) so that devops could decide in a couple of seconds how many VM instances we were going to need, and the likely compute costs - instead of scheduling several meetings where they needed to sort out their internal power struggles of who was right to move the decision forward. I might spend several days (or more) going through code and interviewing developers about how they implemented an authentication protocol so that I can draw it in BAN logic on a slide, which other projects can use to integrate their code with, and our certification/accreditation/infosec people can reason about so the project doesn't get spike stripped at the production gate and delayed a quarter while a post-hoc security analysis gets done on it.

I would meet with our vendors and tech service providers and establish whether they in fact provide the services they said they would, and what that physically means. A great example is whether an o365 tenant supports OAuth2 federated logins with non MSFT tenants or not, and how we are going to enroll users based on the amount of friction the realization of this feature causes(or not). Another one would be pushing standards down into development so nobody would roll their own protocols, and so I could have a short answer to what we implemented. You might see it as ticking a box, but an enterprise is like an airport, and some people get to fly while others don't. The ones who don't are usually stewing over some counterfactual.

I would meet with external consultants, often security and regulatory people, and provide them with the technical details and assurances that would keep them from taking a pound of flesh from the project in the form of a 6 week analysis engagement.

What makes an architect something different is they recognize that when there is direction or demand for a tech project, that creates opportunity for a lot of other very sophisticated technical people to inject themselves into the critical path of the project and use their leverage to extract money, management authority, and other concessions from it. As an architect, you see these people coming, and make sure they do not derail your tech.

I enjoy it because it's solving problems at a higher level of abstraction. I also like doing product, but the architecture urge sabotages that, as in product, solutioning comes at the cost of listening and scaling your listening out to architects to solution stuff.

reply

deedubaya 14 hours ago | prev | next [–]

20% hands on code 20% mentoring

60% facilitating consensus between teams/orgs

reply

loloquwowndueo 12 hours ago | prev | next [–]

Meetings. Endless meetings. reply

tyleo 14 hours ago | prev | next [–]

I'm the software architect for Circuits at Rec Room. Rec Room is a multi-billion dollar UGC-driven online game. Circuits is our in-game programming language. On Monday, Wednesday, and Friday I have large no meeting blocks in my calendar to discourage recurring or pre-planned meetings. Most of this time is spent:

1. Writing code
2. Reviewing code
3. Responding to feedback in our Slack channels
4. Impromptu meetings to help other Engineers get "unstuck" On Tuesday and Thursday I'm open for recurring meetings. Most of this time is spent:
5. In 1:1s with other engineers or team leads
6. In miscellaneous recurring meetings like our "Architects Chat" or "Leadership Meetings" Writing code is *by far* the largest place I allocate my time and also what I would recommend to other engineers who wish to stay on an "individual contributor" track. IMO, if you aren't writing code, you are slowly getting worse at writing code which damages your ability to make recommendations to others. There is no replacement for real experience. Reviewing code is a close second. I review roughly 50 PRs every week which is about 2x what our closest "non-architect" engineers review (usually Engineering Managers). Many of my impromptu meetings and Slack feedback comes from what I see in code reviews. If I leave anything beyond a trivial comment, I like to have time to discuss with the individual to make sure I'm understanding correctly and actually providing useful feedback. Since other people are working in my code-bases, its also important to see that "the ideas are coming together". If PRs into our Circuits code are frequently breaking patterns, its likely a problem with my patterns rather than other engineers.

I don't have documentation on the list as it isn't something I write in an ongoing way. Instead, I prefer to write documentation at the beginning and end of large projects. I don't make a ton of edits as I go because the code is in flux so it can be a waste of time to update both code and documentation. Instead I like to:

1. Start a project with a plan of how everything fits together

```
a. Planning takes anywhere from 1 week to 2 months depending on the size of the project
```

2. Get approval on the plan
3. Write code... if it has to contradict the plan, so be it
4. Update the documentation at the very end so other engineers can use it as a nice reference for the current "state of the world" If anyone is interesting in joining Rec Room, we are hiring :) https://recroom.com/careers?gh_jid=5382144003

reply

FpUser 16 hours ago | prev | next [–]

I was a software architect at the end of the 90s before becoming CTO and then going on my own so it is a bit rusty. Most of my time was spent on high level design of complex business and B2B products and making sure that managers, business analysts and programmers do their jobs. Also selecting technologies etc. I still had about 50% of my time left for actual work so I would always pick some particular chunk and create design documents with diagrams given to programmers and would also code some chunk. I could have avoided doing any "productive" stuff but I did not want to loose touch. Being converted to a pure manager was not in my nature.

reply

clavalle 17 hours ago | prev | next [–]

- First thing in the morning, standups followed by coaching sessions (basically pair programming or code review) for sticky problems Then it varies.

- 1:1s
- project or program (in the sense of both code type programs and long term direction programs) planning or status meetings with various stakeholders, especially POs, PMs, and execs
- digging into particularly difficult or odd code or set up issue that's lead to it being escalated
- meetings with vendors and doing a write up of their product's potential and costs
- creating rough project or program outlines with rough budget estimates for execs
- creating design documents on projects or programs that have a bit of traction
- meeting with other architects to collaborate on wider designs or just share items of interest
- going on HN to tell strangers about my day-to-day life
- create fully functional POCs or reference implementations
- doing post-mortems
- examining code from a potential acquisition or quizzing their engineers
- getting on meetings with clients or potential clients to give them a warm fuzzy that a tech person with an impressive sounding title is listening to show we're taking something Very Sersiously and find them to be Very Important.
- regular Open Office Hours for anyone in the company to drop by and chat about anything (lots of good stuff comes from these -- highly recommend)
- fielding various requests and questions from Security and Ops other teams and whatnot.
- Having What-If meetings with sales or execs (danger! There be dragons here! But, with Sales especially, a really good way to figure out what customers are clamoring for)
- Preparing talks and technical outreach
- Exploring New Shiny Things (though I tend to do this more on off-hours just because there are fewer interruptions and urgencies).

In terms of microservices in particular, I tend to ask a lot of questions about data and work flow, how someone has defined a domain or domains, making sure they're not falling into the myriad anti-patterns, making sure what they're doing is going to be visible, tractable, and play nice with the rest of our services including following our common patterns and idioms (and when I run into idioms that they might not know about I document them and make them easy to find) and not create any unpleasant surprises down the road, and that they have given appropriate thought to evolution of the service and forward and backward compatibility during that process.

Martin Fowler is by far my most influential resource on microservice architectures and doing domain driven design well. But I would strongly suggest literally reading everything you can find on the subject. Mostly because it's many subjects in a trenchcoat. Seeing it all from a few different angles will better arm you for whatever comes your way.

reply

cratermoon 17 hours ago | prev | next [–]

If you flex the title "architect" to include "staff engineer", https://lethain.com/what-do-staff-engineers-actually-do/ By the way, I highly recommend his book Staff Engineer if you're interested in delving deeper.

reply

throw1234651234 18 hours ago | prev | next [–]

I end up doing all the BA / PO / Scrum work and go to a lot of high-level meetings where POs give me some vague requirements they wrote down in 15 minutes between taking their kids to school and soccer practice. Then I cover for the scrum master who takes off on critical release days. Then I go to meetings and track down people to clarify requirements and do an architecture diagram or two and write stories from it and get it "approved" by the PO.

I also pick up stories the team doesn't want, since I feel responsible for them not being clear / easy / broken up enough. In the evenings, I study for cloud exams.

Resources I have found useful is really understanding flowcharts, and the difference between flowcharts and high-level architecture diagrams.

reply

nathias 15 hours ago | prev | next [–]

meetings I imagine reply

johnea 14 hours ago | prev | next [–]

Writing powerpoint with the intention of fucking up the actual developers of S/W... reply

oxff 17 hours ago | prev [–]

Feed stuff to chatgpt, feed it to build pipeline reply