Ask HN: CS papers for software architecture and design?

525 points by avrmav on Nov 25, 2017 | hide | past | web | favorite | 99 comments

Can you please point me to some papers that you consider very influential for your work or that you believe they played significant role on how we structure our software nowdays?

Maro on Nov 25, 2017 [-]

I used to be very interested in Software Architecture, in fact I've read many of the papers cited here.

When I did a startup many years ago, I committed the mistake of paying too much attention to the architecture of the software [1] I was writing, and not enough attention to the product/customer side of it.

The last couple of years I've been de-emphasizing software architecture as an interest, and have been paying much more attention to how product teams build successful products, what the patterns are, etc. I was lucky enough to work at Facebook for a while and got to see (and learn) a very successful working model of product development.

So, while I'm not saying that software architecture is not important (it is), also pay attention to the product/customer side: what choices (software, organizational, hiring, business) allow you to move fast and iterate, to release early and often, to run A/B tests, etc.

I think good software engineers are just as much product guys (and data guys) as they are software guys.

-

[1] https://github.com/scalien/scaliendb

Silhouette on Nov 26, 2017 [-]

I was lucky enough to work at Facebook for a while and got to see (and learn) a very successful working model of product development.

Facebook has been extremely successful commercially, but I think it's dangerous to read too much into how a unicorn like that develops its software, partly because there is survivorship bias at work here, and partly because the most important things Facebook has achieved have relatively little to do with software.

Facebook's golden goose is the network effect. Once it reached a critical mass of users, it was all but unstoppable. Arguably its most impressive technical feat was achieving enough scalability in its infrastructure that it could keep up with that many users and that much data. That was a remarkable success story by any standard, but while it surely has a software element, no doubt it involved much more than just code.

On the other hand, to a first approximation Facebook has had infinite resources for most of its existence. It operates in an online environment where problems can be fixed even in production. And it doesn't really do anything that is going to cause catastrophic, unfixable consequences if something does break for a while. That is a list of luxuries that few software development teams enjoy, and what works if you can make those assumptions won't necessarily be a good idea if you can't.

As you say, you do have to pay attention to other factors as well, but there are not many organisations that have as much room to manoeuvre on the software side as Facebook does.

Maro on Nov 26, 2017 [-]

When you're looking at templates for success, then looking at the successful companies (survivorship bias) is a good start.

The fallacy is when it turns out that there are lots of other companies doing the same thing, but those companies aren't successful. This is the problem with all those blog posts examining the daily habits of highly successful people (like, they wake up early, or, they eat energy bars to save time, etc). Clearly, just doing these things won't make you a billionaire.

For what I wrote, I believe this doesn't apply, because of the stark difference between the culture I experienced at FB and other companies. (Admittedly, this is not a stat.sign. sample.) So in my experience it's _not_ true that other companies are doing the same thing...

Having said that, you're right that FB is a one in a million company, and probably nobody reading this will be the next M.Zuckerberg... But still, if you want to be the best in your domain (eg. best todo app), I strongly think these are good patterns to follow.

Silhouette on Nov 26, 2017 [-]

The fallacy is when it turns out that there are lots of other companies doing the same thing, but those companies aren't successful.

Well, there is no shortage of web startups trying to be all agile and fast-moving that still fail, and among the ones that survive long enough to become established, there seem to be plenty of problems with reliability and security issues that better software design might have avoided.

Facebook seem to have significant problems quite often too. I've seen teams redistribute their planned Facebook spending for entire ad campaigns across other channels, because the Facebook UI for setting up the ads was so broken on that day that it was impossible to run the intended FB campaign, or because FB's approval system for promotional content rejected something for obviously incorrect reasons. In one case, FB made a rookie mistake in their payment processing that stopped everything to do with Facebook ads dead for that business until the problem could be resolved, which itself was only possible because of some personal contacts who happened to work at Facebook and could escalate the issue internally.

The thing is, if you're Facebook, you can survive repeatedly causing this sort of hassle for the people paying your bills, because you're big enough that they'll probably come back and try again another time anyway. It's still x% of your potential revenue that you're throwing away, but you don't need that revenue to remain a viable business. However, if you're almost anyone other than Facebook, those kinds of quality control issues will damage your reputation and ultimately sink your business if they become serious enough.

Clubber on Nov 25, 2017 [-]

Agreed, good architecture doesn't come into play until you need to add a lot of complex features or scale significantly. Initially architecture doesn't really mean squat. I would concentrate on making the codebase flexible, but that's about it. I've regretted making some of my software with a cool but complex architecture when I should have been focused on what the product does.

humanrebar on Nov 25, 2017 [-]

> I would concentrate on making the codebase flexible, but that's about it

Arguably that's the property of good architecture. Standardizing your entire codebase also works, but only as long as those initial standards stay smart (1), and even then, that relies on discipline, tools, and incentives that are typically not all in place.

(1) A standard might be "all webservers will be written in Java 6". The benefits of choosing that standard tend to be front-loaded with a gradual decline into net-negative with no affordable path to a better standard.

qznc on Nov 25, 2017 [-]

To design your software perfectly requires to know the future. You want the things that will change flexible and the rest simple. In practice, you will not hit the sweet spot and either over-engineer too much or hardcode too much (or both in different parts).

gregmac on Nov 26, 2017 [-]

I tend to place the most emphasis on loose coupling, and being very strict about interfaces between components -- both in terms of documentation and addition/changes. Ideally documentation includes unit or integration tests. This applies to separate high level services as well as it does to internal code structure (classes/modules/etc).

Well documented and tested interfaces of a component make that component easy to replace or rewrite if needed.

Nearly anytime I have worked on or been involved with a project where I or someone else tried to design specifically for a vague future requirement that wasn't entirely clear or guaranteed, it didn't work out: either the requirement never happened, or was so materially different that what we wrote was wrong. As a result, the code we wrote went unused or worse, just got in the way and ended up as unnecessary technical debt.

madisonmay on Nov 25, 2017 [-]

In Knuth's words: "Premature optimization is the root of all evil". Although I guess in this context, you're emphasizing that it's more valuable to solve the right problem poorly than it is to solve the wrong problem well.

Clubber on Nov 26, 2017 [-]

I didn't say anything about solving the right problem poorly, but you are probably speaking a truth, depending on what poorly means. It certainly isn't ideal. I've seen solutions designed so poorly that when it came time to add features that paying clients wanted, it was significantly more difficult (significant risk) than if it was designed well.

I'd say solve the right problem well, without spending time on requirements you don't yet have. These would include a large scalability requirement and over-modularization, like breaking out 10 "micro services."

kenver on Nov 26, 2017 [-]

There’s a big difference between being tunnel visioned on architecture and not getting anything done, and the very common scenario of not seeming to care at all.

Having been a contractor at many companies Im pretty sick and saddened of seeing how common it is for every layer of the product to be implemented as a global state Singleton.

thewarrior on Nov 25, 2017 [-]

Could you elaborate on the insights you gained ?

At least on a high level . It would help a lot of people here .

Maro on Nov 25, 2017 [-]

I think most of the things people know, it's covered in books like Lean Startup, etc.

It's a bit like eating well and working out. We all know we should do it, but most people don't actually eat well and work out. Then, when you see somebody who does it and looks great, you ask them, "What's your secret?". But it's not a secret, it's just that most people don't do it, because it's hard :)

One story: When I was at FB, I happen to know that a team of size S conducted X experiments in 6 months (I can't disclose the number). As it happens, I have worked in similar size teams in other companies, and there the number was ~X/20, and sometimes 0. It don't matter how good your software architecture is, if you're trying out just 1 thing instead of 20 things... I call this velocity.

Another good story: many semi-successful companies end up in a place where there is an initial product/software which gets them a lot of growth, and then ~5 years into the startup, the bigger, more mature, bigger team decided that the old legacy code is holding them back, and they're going to REWRITE IT. Maybe in some fancy new language, or a fancy new architecture like micro-services. The estimate is 6-9 months to get to first light. But it will probably end up taking 3-5 years, because the legacy had a lot of fine-tuning in it, and it turns out many of the problems are hard to fix, moving the production to a new thing is REALLY HARD, and all those fancy new technologies are actually far from perfect, plus the current team doesn't have a lot of experience with it. Compare to this what Facebook did with its PHP codebase: at some point it became a bottleneck (the crappy language and the runtime speed), but there was a never a from-the-ground-rewrite. Instead they ended up writing several iterations of better runtimes, and since they were changing the runtime anyway, they "fixed up" the language (while keeping it mostly backward compatible). The new language is called Hack and the new runtime is called HHVM. The cool thing is, in all this time, there wasn't a rewrite, so they were able to keep shipping new features on Facebook, run A/B tests, iterate on the product. Compare this to one of the companies I worked at, where they did a rewrite , and now customers have to chose between the old and a new thing, it's not transparent, because <software issues>. There's a book about Hack/HHVM, one of the Facebook guys wrote it, iirc the first chapter is about this whole story. See this blog post for links: http://bytepawn.com/hack-hhvm-second-system-effect.html

In general, the principles I've seen to work really well:

- write good code, but don't do big rewrites

- cont. delivery: always be shipping to master and production in small increments (don't have big git branches that aren't in production, at the end it will be scary to merge it and put it into production)

- cont. integration: have tests and run them on every commit (you probably need some testing nazis to enforce this...)

- if you (your team) can't write a good monolith, you (your team) also can't write a good MSA

- invest heavily in linting and other automated ways to catch and conform code when it is committed

- programming language doesn't matter that much, just pick one for each domain (eg. web, mobile, etc), and stop thinking about it, and don't let people waste their time on arguing over it too much; instead invest heavily in tooling that supports all the other aspects (like code reviews, perf, experimentation, etc)

- 1 other person should review and okay the code before it goes into production

- aggressivley remove obstacles from shipping stuff to production

- make it easy to run experiments, and run a lot of experiments

- don't hire people who just want to write code, or think their job stops there

quotemstr on Nov 25, 2017 [-]

> When I was at FB, I happen to know that a team of size S conducted X experiments in 6 months (I can't disclose the number). As it happens, I have worked in similar size teams in other companies, and there the number was ~X/20, and sometimes 0. It don't matter how good your software architecture is, if you're trying out just 1 thing instead of 20 things... I call this velocity.

Exactly! FB has the most productivity per developer of any company I've seen, and this efficiency is a direct result of management systematically lowering barriers to writing code and investing in developer productivity tools. Other companies have elaborate rules and procedure and committees and style guides built around *stopping* code. Facebook encourages writing code.

(And before you say: no, the codebase doesn't take a quality hit. Turns out that this ceremony around writing code is just unnecessary, contrary to population opinion.)

Silhouette on Nov 26, 2017 [-]

(And before you say: no, the codebase doesn't take a quality hit. Turns out that this ceremony around writing code is just unnecessary, contrary to population opinion.)

Which part of the codebase? Facebook's back-end systems do a very impressive job given the scale involved. On the other hand, its front-end systems appear to be mediocre in most important respects. A business without Facebook's advantages that wrote a UI as slow and buggy as Facebook's main or advertiser UIs often are could be in serious trouble.

chubot on Nov 25, 2017 [-]

I agree with all of this for products. To a first approximation, you want to bump your head into reality as often as possible.

But I would say software architecture matters more for infrastructure like storage systems, operating systems, and programming languages than it does for products. I think a lot of the literature on software architecture is about those domains.

Those things are harder to do iteratively. And language choice matters more in those domains.

Programming is a huge field now, and a lot of choices are domain-specific. Including how much you should care about architecture and programming languages.

Maro on Nov 25, 2017 [-]

You're probably right! But, most software engineers aren't writing a compiler/database/OS, and some of the ones who are, probably shouldn't :)

ChuckMcM on Nov 25, 2017 [-]

I had a similar journey although came away from that point headed in a slightly different direction. As you have studied the processes by which software is developed have you noticed a correlation between developing pieces that are architecturally important (many things are touched by the choices) and planning investment by the team?

Maro on Nov 25, 2017 [-]

What I've seen is that most orgs, once they reach a certain size (100-200 people), start to spend a significant time planning. Part of this is human nature: once there are enough people, there will be managers, and plans are something managers show their hire ups to prove they're in control and doing a good job. But, I don't want to sound cynical, clearly planning is useful.

I used to think that spending too much time on planning is a waste of time , but then I noticed that teams at Facebook also spend a lot of time at the beginning of each half (6 mo) doing planning. However, I wasn't able to pick up tricks or patterns that made FB good at this, other than obvious/useless things like all the people making the plans were really smart / domain experts. I did notice that the plans/planning process was very lose, there was no methodology, often the outcome was a bunch of bullet points. And they were very conscious about the value of the plan: they knew that if things go well and they move quickly, probably a lot of things will change in 3-4 months, and they will diverge from the plan. The value is thinking things through, agreeing on goals, spinning up teams, building things (eg. dashboards) to track progress, etc.

Another story about the value of planning: at another company we hired a manager, and one of the things this guy did was introduce Capacity Planning. At the core it's a simple thing: make a big matrix, where the rows are the projects people want to work on, and the columns are the available people/teams. Collect all the projects, and make people put down their estimates for each (requires X time from team Y, etc). The value is, it's a very explicit way to show what people want to work on, obviously it will be more than you can actually do. So then you can make a very explicit choice about what you're doing and what you're not doing in the next X months. Back when we did this, there was an interesting insight the first time around: since we could count man days, we learned we're spending iirc ~70% of our time keeping the lights on (infra, etc), and only ~30% working on new things.

But despite this insight, I thought it's a waste of time, it's too slow/formal/sluggish. But then I saw FB also spends significant time on planning (although it's much less formal, because they can get away with it, for a long list of reasons), so I thought okay, spending time planning is probably the way to go.

Then I went to a company which was suffering from a focus problem. Every 2 weeks they'd have a prioritization meeting, where projects would get re-prioritized, which leads to obvious problems. Here I realized, for this problem, Capacity Planning would be a good thing, because one of the things it gets you is a long(er) term commitment to certain projects.

Overall, my takeaway is to spend 10-20% of time on planning and related meta-activities. Be conscious that the value is in thinking things through etc, and the actual path may diverge, but that's okay. Depending on the culture, there are a number of things that planning gives you, eg. a commitment to work on X for Y time. Make metrics and dashboards to track things, but don't overdo it.

gobengo on Nov 25, 2017 [+17]

davedx on Nov 25, 2017 [-]

For me, the most influential was "Out of the Tar Pit".

From the abstract: "Complexity is the single major difficulty in the successful development of large-scale software systems. Following Brooks we distinguish accidental from essential difficulty, but disagree with his premise that most complexity remaining in contemporary systems is essential."

[1] https://github.com/papers-we-love/papers-we-love/blob/master...

charlysl on Nov 26, 2017 [-]

I am not an expert and I might be well be wrong, but I decided to stop reading this paper after page 28 because I thought it is, well, not serious research and IMHO BS. I decided then to lookup the authors in google and scholar, and came up just with this paper, no other info or any academic background as far as I can tell; there is a Ben Moseley from Washington Uni, but his paper is not listed among his papers so I assume it is a different Ben.

I would have expected the authors of a paper that makes such revolutionary and sweeping claims to have more of a trail.

But of course, you have to judge a paper by its content.

The reason I stopped is because they quote heavily from a well respected book (often described as an updated SICP) that I am studying now, "Concepts, Techniques and Models of Computer Programming", and my interpretation of what the book says is at odds from how they apply it in the paper.

For instance, in the book the difference between a formal specification and an informal one is not in precision, but in that the formal one uses a mathematical language. However, in the paper it says that a formal specification is the same as formal requirements (synthesised by the engineer), which are different from informal ones from the user. These definitions can't both be right.

They also claim that in the ideal world control (basically order) can be entirely omitted. But what if from the user's informal requirements we must deduct that there are events that the user expects to be ordered?

The paper also claims that concurrency is not relevant in an ideal world, given that all operations are instantaneous. This would be impossible if there indeed there would be essential control, given that parallelism in the world is inescapable.

At this point I was convinced that I was wasting my time, and now I know why this paper hasn't had any impact in the mainstream, as one commentator has wondered.

fermigier on Nov 25, 2017 [-]

I agree that's a wonderful paper to read, but has it had significant influence in the mainstream?

chubot on Nov 25, 2017 [-]

I read this more than 5 years ago, I think from a recommendation on HN. But I must have gotten something different out of it than most people.

It is advocating a particular architecture. But that architecture is essentially LAMP, as far as I can tell. It's what we ALREADY do!!!

From the paper:

FRP is currently a purely hypothetical approach to system architecture that has not in any way been proven in practice. It is however based firmly on principles from other areas (the relational model, functional and logic programming) which have been widely proven.

In FRP all essential state takes the form of relations, and the essential logic is expressed using relational algebra extended with (pure) user- defined [18] functions.

[18] By user-defined we mean specific to this particular FRP system (as opposed to pre- provided by an underlying infrastructure)

- *All essential state takes the form of relations* This is a database. (SQL deviates from the relational model, but I don't view that as important here. An SQL database stores everything as relations.)

- *Logic is expressed using pure user-defined functions* This is PHP / CGI / FastCGI. PHP is imperative, but the entire program is a pure function, because the request state is cleared between every request.

What am I missing? I'm being totally serious -- this is what I got out of it when I read it 5 years ago.

You can quibble with the details of PHP or Rails not being pure functions, but I believe what's important is the architecture, not how the source code looks. The *essential state* is in the database, in the form of relation. Accidental state is thrown away.

TL;DR -- PHP/MySQL is functional and relational.

DenisM on Nov 25, 2017 [-]

FRP is a constrained environment, which makes it easier to reason about. PHP is comparatively unconstrained, so when reading code you can’t make as many shortcut assumptions. Of course you could write in PHP in accordance to the rules, but subsequent maintainers, including the future you, cannot trust this self-imposed discipline. In other words, you wouldn’t argue that assembler is just fine because through vigor and discipline you can build PHP-like application. Similarly, PHP is not well-suited to stand in for FRP.

chubot on Nov 25, 2017 [-]

I probably shouldn't have said PHP, since people have reactions to that language that may confuse the issue.

I should have said stateless Python front ends. Not even CGI/FastCGI, but just plain HTTP front ends. Many large websites use this architecture (YouTube, Instagram, etc.)

An example of something that doesn't follow the architecture is a stateful node.js or Go server.

I would say that the typical web architecture is similar to what they are talking about, with the benefit of existence :)

But you're right in the sense that they are trying to be more strict, starting on page 50:

- benefits for state -- avoid useless accidental state. This is the same philosophy behind SQL.

- benefits for control -- They are being more strict here, but I think it is missing a lot, because sometimes you need control flow.

- benefits for code volume -- I would need to see a real system to evaluate this claim. It's not fair to compare systems that exist with ones that don't :)

- benefits for data abstraction -- SQL agrees here. You don't abstract data. People sometimes make this mistake in their OOP languages, but's incidental.

DenisM on Nov 26, 2017 [-]

I see what you mean. I was thinking of smaller granularity, less than an entire HTTP request. You have a good point.

charlysl on Nov 26, 2017 [-]

"All essential state takes the form of relations This is a database" - relations are a more abstract concept than this, see relational programming

"Logic is expressed using pure user-defined functions This is PHP / CGI / FastCGI. PHP is imperative, but the entire program is a pure function, because the request state is cleared between every request." - the entire program is not a pure function as seen by clients; to be functional, sending an identical request would always have to display the same information, but it won't if the page was updated between requests. Functional means that a given input (the request in this case) would always return the same output (page), just like a function in the mathematical sense would.

"PHP/MySQL is functional and relational." In a behavioural view of "functional", what matters is whether an operation always produces the same output for a given input; this is certainly possible with imperative languages, with some discipline. We can't apply a definitional view in this case, because PHP is multiparadign, supporting both stateful and functional programming , so you can't write operations that are functional by definition, you have to know what your are doing.

BTW, I haven't read the paper yet, I guess it will change my views after I do

monocasa on Nov 25, 2017 [-]

FRP is like excel or well constructed makefiles in that the runtime understands the data flow. Then on updates to any data, you walk through the dataflow graph, running pure functions to update each node.

chubot on Nov 25, 2017 [-]

I think you're talking about functional reactive programming, which they are not talking about. See page 42:

[16] Not to be confused with functional reactive programming [EH97] which does in fact have some similarities to this approach, but has no intrinsic focus on relations or the relational model

davedx on Nov 25, 2017 [-]

Maybe not in the mainstream, but I do see echos of the philosophy of reducing accidental complexity in clojure, elm, react/redux, rust.

anandabits on Nov 25, 2017 [-]

Not yet, but I suspect that is partly the mainstream not catching up to the ideas it contains yet.

chubot on Nov 25, 2017 [-]

See my sibling comment -- I honestly believe it's describing what people already do. I'm curious what someone who's read the paper (like I have) thinks.

fabuzaid on Nov 25, 2017 [-]

I'm surprised no one has mentioned the David Parnas papers: "On the Criteria To Be Used in Decomposing Systems into Modules" is an all-time classic [1]. Much more philosophical than most of the suggestions here so far, but this paper really drove home to me the importance of abstraction in software design. (Also covered in the Morning Paper [2].)

[1] https://www.cs.umd.edu/class/spring2003/cmsc838p/Design/crit.... [2] https://blog.acolyer.org/2016/09/05/on-the-criteria-to-be-us...

jonjacky on Nov 25, 2017 [-]

An excellent textbook that teaches Parnas' approach is "Software Engineering: Planning for Change" by David Alex Lamb. The chapter Detailed Design is especially pertinent to this thread. The book is from 1988. I once wrote the author to ask if there would be another edition. He replied, "publishers aren't interested because it isn't object oriented". In fact, its lessons are as relevant to object-oriented programs as any others.

jonjacky on Nov 25, 2017 [-]

Another excellent Parnas paper: "A Rational Design Process: How and Why to Fake It"

zaptheimpaler on Nov 25, 2017 [-]

DynamoDB paper [1] - one of the big AP, KV stores that kicked off the NoSQL wave.

Paper about Akamai[2] - "Nuggets in content delivery"

Also, acolyers blog "the morning paper". He reviews and explains one paper every day. its great because you get a sense of whats current.

[1]http://www.allthingsdistributed.com/files/amazon-dynamo-sosp...

[2]https://www.akamai.com/cn/zh/multimedia/documents/technical-...

elvinyung on Nov 25, 2017 [-]

nit: Dynamo != DynamoDB: http://www.allthingsdistributed.com/2012/01/amazon-dynamodb....

godelmachine on Nov 25, 2017 [-]

The Morning Paper's Adrian Colyer is a human Machine. I religiously read his papers.

sriram_malhar on Nov 25, 2017 [-]

Hints for Computer System Design, Butler Lampson. https://www.microsoft.com/en-us/research/wp-content/uploads/...

End-to-End Arguments in System Design, http://web.mit.edu/Saltzer/www/publications/endtoend/endtoen...

I think Rich Hickey (of Clojure) makes lovely points about application system design. I know you are looking for papers, but Rich's talks have influenced me greatly of late. https://github.com/tallesl/Rich-Hickey-fanclub

EdwardCoffin on Nov 25, 2017 [-]

Lampson has done an update of his hints paper, in the form of a presentation [1] (the video of the talk), [2] (the slides).

[1] http://www.heidelberg-laureate-forum.org/blog/video/lecture-...

[2] http://bwlampson.site/Slides/Hints%20and%20principles%20(HLF...

Edit: added link to slides

godelmachine on Nov 27, 2017 [-]

I tried reading Hints for Computer System Design, Butler Lampson - but failed to make any sense of it. Couldn't place it in neither hardware or software. May I ask you to share your key takeaways from the paper? Sincerely,

mindvirus on Nov 25, 2017 [-]

Here are some books, talks and papers that I've found really influential:

Clean Architecture (https://www.amazon.com/dp/0134494164) - Uncle Bob has been talking about this for years, but it's a really good exploration of how to build systems.

Turning the Database Inside Out is a talk that really made Apache Kafka and stream processing click with me. https://www.confluent.io/blog/turning-the-database-inside-ou...

Going through and implementing the Raft algorithm was also very formative for me - it's the first time I really started to grok the challenges with building large scale distributed systems. https://raft.github.io/

And to add a paper to the list to not totally go off topic - Birrell's paper "An Introduction to Programming with Threads" I thought was a very useful read - in part for the historical context, but he also breaks down many fundamental concepts that will look very familiar to the modern reader (it was written nearly 30 years ago). It's also very readable. https://birrell.org/andrew/papers/035-Threads.pdf

einarvollset on Nov 25, 2017 [-]

In general, CS (the science) does not often concern itself with the kind of architecture decisions day-to-day programmers deal with. Instead, the field tends to offer architecture advice in particularly hairy areas (DB design, distributed FT consensus, etc). In my opinion, that’s how it should be - the goals of academics are very different to those shipping code in the real world. A great developer will be aware of when to turn to academia and when to turn to industry best practice - too often developers are not aware that they are wondering into a hairy area, and end up re-inventing flawed and inefficient versions that are well understood.

drharby on Nov 25, 2017 [-]

That last line of yours is so very true. Whenever i find myself going down the rabbit hole i pick up the nearest relevant academic text and just speedread the relevant chapter to tailor my search doe relevant readymade solutions.

Developer life is like googlefu xtreme edition

capkutay on Nov 25, 2017 [-]

i was going to make a similar comment but this sums it up very well.

CS papers work best when dealing with very specific, nitty gritty areas that require research after defining a hypothesis or problem to solve.

'Software architecture' is fairly broad and papers on those topics would read more like sociology papers than CS papers (e.g. how do people write code? here are some examples, here are some general patterns people like etc).

Looking back on reading that sort of stuff, it was nice but I very rarely apply it in day to day software development compared to readings that are more niche to the problem I'm solving that week.

mtrn on Nov 25, 2017 [-]

I liked reading: Program design in the UNIX environment by Rob Pike. Sometimes, when I think about which features to add to a program I step back to see, whether a combination of program would actually yield better results. A practical view into what orthogonality could mean in the tooling world.

[1] https://nymity.ch/sybilhunting/pdf/Pike1983a.pdf

igolden on Nov 25, 2017 [-]

Awesome recommendation

santix on Nov 26, 2017 [-]

Good one.

At the bottom of page 3 it says that on UNIX V7 they added an *unbuffered* (-u) option to *cat* and then removed it on V8. Does anybody know why they removed it?

I checked out the man for the GNU version and it says

```
  -u     (ignored)
```

elvinyung on Nov 25, 2017 [-]

I think one of the most useful papers about scaling modern web architectures is Armando Fox and Eric Brewer's *Harvest, Yield, and Scalable Tolerant Systems*: https://pdfs.semanticscholar.org/5015/8bc1a8a67295ab7bce0550...

Not only was it the paper that first put into written word the CAP theorem, but it also prescribes design principles for scaling out applications with graceful degradation and "orthogonal decomposition" (i.e. service-oriented architectures).

It's also a fairly short and easy read at only 4 pages, and I'd definitely recommend everyone interested in practical modern distributed systems to take a look.

qznc on Nov 25, 2017 [-]

Conway's Law [0] is an essential aspect for large scale software development:

> Any organization that designs a system (defined broadly) will produce a design whose structure is a copy of the organization's communication structure.

[0] http://www.melconway.com/Home/Conways_Law.html

godelmachine on Nov 27, 2017 [-]

Nice point. Will keep that in mind.

DyslexicAtheist on Nov 25, 2017 [-]

I compiled a list some time ago, I'm still actively updating it (if you have ideas for work that should be included please let me know):

"An incomplete list of classic papers every Software Architect should read" https://blog.valbonne-consulting.com/2014/06/09/an-incomplet...

chatmasta on Nov 25, 2017 [-]

Classic reference is Architecture of Open Source Applications [0]

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

Kagerjay on Nov 25, 2017 [-]

thats neat there's a full spreadsheet application on there too in this jsfiddle http://jsfiddle.net/audreyt/LtDyP/. No external libraries used either, 99 lines

jph on Nov 25, 2017 [-]

LISP. Recursive Functions of Symbolic Expressions Their Computation by Machine by John McCarthy at MIT.

https://aiplaybook.a16z.com/reference-material/mccarthy-1960...

"Introduction: A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system called the Advice Taker, whereby a machine could be instructed to handle declarative as well as imperative sentences and could exhibit "common sense" in carrying out its instructions."

lachenmayer on Nov 25, 2017 [-]

Not sure it says a lot about 'architecture', but in terms of 'design', Daniel Jackson's "Rethinking Software Design" has been an absolute eye-opener: https://www.youtube.com/watch?v=cNe6g0qczxE

The premise of the talk is that while software engineering goals & processes are well-defined and well-understood, the same does not apply for software design. He sets up a framework of 'purposes' and 'concepts', which in an ideal design should be a 1:1 mapping. He points out many examples of 'design smells', where this 1:1 mapping is violated, for whatever reason (engineering reasons, product management failures, etc.)

Definitely useful not just to developers, but anyone involved in writing software/affected by the process of writing software.

charlysl on Nov 26, 2017 [-]

I am a big fan of Daniel Jackson's approach to teaching programming, and really like his courses, in particular the mindblowing "Elements of Software Construction" [1], which is, by a mile, the best programming course I have come across, and which I believe is sadly underrated; I know there are more recent versions of this course, by a different professor, that use the more popular (at least in this forum) Python instead of Java, but IMHO with all due respect, they are a bit dumbed down and I believe that you won't get the same out of them. It is not really a Java course, it has improved my programming (and, very specially, design) in general, in any language. It completely changed the way I look at design patterns, for instance; it made me really understand functional programming and state machines; what OO really is about; how to choose a software paradigm for a given problem; and now I use JSP stream processing to model many programs (nothing to do with java JSP, it was created by his dad in the 70s, "the other Michael Jackson").

To really appreciate how much thought went into the design of this course, I recommend reading "A New Approach to Teaching Programming" [2]

As for the Concepts and Purposes you refer to, there is more on this in [3] [4] [5] in the MIT 6.170

[1] https://ocw.mit.edu/courses/electrical-engineering-and-compu...

[2] http://people.csail.mit.edu/dnj/articles/teaching-6005.pdf

[3] https://stellar.mit.edu/S/course/6/fa16/6.170/courseMaterial...

[4] https://stellar.mit.edu/S/course/6/fa16/6.170/courseMaterial...

[5] https://stellar.mit.edu/S/course/6/fa16/6.170/courseMaterial...

markc on Nov 25, 2017 [-]

Kreps and Kleppmann on stream processing, log abstraction, event sourcing:

https://engineering.linkedin.com/distributed-systems/log-wha...

https://martin.kleppmann.com/2015/03/04/turning-the-database...

https://www.confluent.io/blog/making-sense-of-stream-process...

jesperlang on Nov 25, 2017 [-]

A lot of good suggestions, but quite technical. Are there any good ones focusing on the design process, with a bias towards CS?

I found this very intriguing:

Towards a Theory of Conceptual Design for Software
https://groups.csail.mit.edu/sdg/pubs/2015/concept-essay.pdf

fermigier on Nov 25, 2017 [-]

"Scripting: Higher- Level Programming for the 21st Century" by J. Ousterhout was certainly very influencial (cf. Python, Ruby, etc.)

https://web.stanford.edu/~ouster/cgi-bin/papers/scripting.pd...

Not sure the paper in itself aged so well.

Also, see:

https://en.wikipedia.org/wiki/List_of_important_publications...

charlysl on Nov 26, 2017 [-]

Thx for the wikipedia link, this is probably the best paper list I've seen so far

tonyspiro on Nov 25, 2017 [-]

The folks at Heroku have the 12 Factor App https://12factor.net

q3k on Nov 25, 2017 [-]

Oldie but goodie: Large-scale cluster management at Google with Borg [1]. Kicked off the mainstream idea that maybe (dev)ops shouldn't care about servers and Linux service configuration or where exactly their code runs.

[1] - https://static.googleusercontent.com/media/research.google.c...

charlysl on Nov 25, 2017 [-]

[1] Joe Armstrong 2003. "Making reliable distributed systems in the presence of software errors" (the Erlang paper, perhaps the most famous implementation of message passing concurrency).

[2] John Backus 1979. "Can programming be liberated from the von Neumann style? A functional style and its algebra of programs." (a famous case for functional programming)

[3] Victor R. Basili and Albert J. Turner. 1975 "Iterative enhancement: A practical technique for software development" (Agile in 1975? Actually it seems iterative development can be traced back to the 50s)

[4] Andrew D. Birrell and Bruce Jay Nelson 1984 "Implementing remote procedure calls" (The idea of making a distributed language operation similar to a local language operation)

[5] Edsger W. Dijkstra. "Go To statement considered harmful" (the mother of all those "considered harmful" titles)

[6] Carl Hewitt 1977. "Viewing control structures as patterns of passing messages". (seminal study of asynchronous messaging)

[7] Carl Hewitt, Peter Bishop, and Richard Steiger 1973. "A universal modular ACTOR formalism for artificial intelligence" (idem)

[8] Charles Antony Richard Hoare 1974. "Monitors: An operating system structuring concept". (an early refinement of a very influential concurrency concept)

[9] Charles Antony Richard Hoare 1979. Communicating sequential processes (an early presentation of invariants for state machines)

[10] deleted

[11] Gilles Kahn and David B. MacQueen 1977. "Coroutines and networks of parallel processes" (non preemptive concurrency, an alternative to threads for collaborative concurrency)

[12] Robert A. Kowalski 1979. "Algorithm = logic + control". (A logic program can be “read” in two ways: either as a set of logical axioms (the what) or as a set of commands (the how); the idea behind relational programming, like in prolog)

[13] Nancy Leveson and Clark S. Turner 1993. "An investigation of the Therac-25 accidents." (the mother of all race conditions papers? Focuses the mind about the dangers of interleavings)

[14] Henry Lieberman. 1986 "Using prototypical objects to implement shared behavior in object- oriented systems." (influenced js? Understanding delegation)

[15] George A. Miller. 1956 "The magical number seven, plus or minus two: Some limits on our capacity for processing information." (this might be pushing it, but maybe the original case for managing complexity)

[16] Chris Okasaki. 1998 "Purely Functional Data Structures." (how to design functional algorithms)

[17] John C. Reynolds 1975. "User-defined types and procedural data structures as complementary approaches to data abstraction." (the fundamental abstract data types concept)

[1] http://erlang.org/download/armstrong_thesis_2003.pdf

[2] https://www.thocp.net/biographies/papers/backus_turingaward_...

[3] https://www.cs.umd.edu/~basili/publications/journals/J04.pdf [4] http://www.bighole.nl/pub/mirror/www.bitsavers.org/pdf/xerox...

[5] https://homepages.cwi.nl/~storm/teaching/reader/Dijkstra68.p...

[6] https://www.cypherpunks.to/erights/history/actors/AIM-410.pd...

[7] http://worrydream.com/refs/Hewitt-ActorModel.pdf

[8] http://www.cs.unm.edu/~cris/481/hoare-monitors.pdf

[9] http://lass.cs.umass.edu/~shenoy/courses/677/readings/Hoare....

[11] https://hal.inria.fr/inria-00306565/PDF/rr_iria202.pdf

[12] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.472...

[13] https://www.cs.umd.edu/class/spring2003/cmsc838p/Misc/therac...

[14] http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.6...

[15] http://www.sns.ias.edu/~tlusty/courses/InfoInBio/Papers/Mill...

[16] https://www.cs.cmu.edu/~rwh/theses/okasaki.pdf

[17] https://pdfs.semanticscholar.org/71db/39fe3e2a6b80c52b18bd24...

ramchip on Nov 26, 2017 [-]

+1 for "Making reliable distributed systems in the presence of software errors". It's a long read, but it has completely changed the way I approach programming.

avrmav on Nov 25, 2017 [-]

owo, thx a lot for the effort.

fabiopetrillo on Nov 25, 2017 [-]

Philippe Kruchten's The 4+1 view model of architecture is a very influential work for software architecture/design: https://pdfs.semanticscholar.org/450b/1e34bc072589ed83f72f6f... .

cpeterso on Nov 25, 2017 [-]

Eric Evans' *"Domain-Driven Design: Tackling Complexity in the Heart of Software"* book is long and pretty dry, but it's immediately practical and opened my mind to how object-oriented design is supposed to work.

https://amzn.com/0321125215

https://en.wikipedia.org/wiki/Domain-driven_design

darethas on Nov 25, 2017 [-]

Is there a world of software architecture that exists outside the "OOP" patterns we have seen in the last 2 decades? A quarter of the way through my career and I have already grown weary of OOP. It's promises are never realized. It is full of zealots who can argue with you for ages but don't deliver very much on real world, performant software.

osullivj on Nov 25, 2017 [-]

Brad Cox: Planning the Software Industrial Revolution. A classic from the creator of Objective-C. Incredibly far sighted in 1990, and still very relevant today.

[1] http://bat8.inria.fr/~lang/hotlist/free/licence/papers/cox/C...

tuukkah on Nov 25, 2017 [-]

An open graph visualization system and its applications to software engineering: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.106....

yogeshp on Nov 26, 2017 [-]

Here are some links (papers & talks) on scalable software architecture

https://github.com/Developer-Y/Scalable-Software-Architectur...

eternalban on Nov 25, 2017 [-]

> .. how we structure our software nowdays

The current trends were mostly blog driven. (Notable exception being the design approach to *distributed systems and databases*, which has been noted by many in this thread. Brewer's CAP theorem, etc., of course, were hugely influential.)

In terms of "CS" papers, there were a few that argued *against* the then prevailing canon such as object orientation. In terms of *theory*, the theoretical foundation of much of the current approach can be found (surprisingly) in papers from the 70s and 80s, e.g. Hoare's CSP (which influenced Go) is circa '85.

westurner on Nov 25, 2017 [-]

"The Architecture of Open Source Applications" Volumes I & II http://aosabook.org/en/

"Manifesto for Agile Software Development"
https://en.wikipedia.org/wiki/Agile_software_development#The...

"Catalog of Patterns of Enterprise Application Architecture"
https://martinfowler.com/eaaCatalog/

Fowler > Publications ("Refactoring ",)
https://en.wikipedia.org/wiki/Martin_Fowler#Publications

"Design Patterns: Elements of Reusable Object-Oriented Software" (GoF book)
https://en.wikipedia.org/wiki/Design_Patterns

.

UNIX Philosophy https://en.wikipedia.org/wiki/Unix_philosophy

Plan 9 https://en.wikipedia.org/wiki/Plan_9_from_Bell_Labs

Distributed Systems

CORBA > Problems and Criticism (monolithic standards, oversimplification,):
https://en.wikipedia.org/wiki/Common_Object_Request_Broker_A...

Bulk Synchronous Parallel: https://en.wikipedia.org/wiki/Bulk_synchronous_parallel

Paxos: https://en.wikipedia.org/wiki/Paxos_(computer_science)

Raft: https://en.wikipedia.org/wiki/Raft_(computer_science) #Safety

CAP theorem: https://en.wikipedia.org/wiki/CAP_theorem

inopinatus on Nov 25, 2017 [-]

Not a paper, but Martin Fowler's book "Refactoring" changed the way I structure my domain models forever.

justonepost on Nov 26, 2017 [-]

Decouple and cohesion. Beyond that it's pretty domain dependent.

JeroenRansijn on Nov 25, 2017 [-]

Write code that is easy to participate in.

frantzmiccoli on Nov 25, 2017 [-]

I would recommend Martin Fowler articles https://martinfowler.com/

harishneit on Nov 25, 2017 [-]

For me, it would be:

1. actor model of computation (https://arxiv.org/abs/1008.1459) 2. map reduce (https://research.google.com/archive/mapreduce.html)

macintux on Nov 25, 2017 [-]

On the industrial side, Jim Gray's paper complements Hewitt's nicely.

http://www.hpl.hp.com/techreports/tandem/TR-85.7.pdf

(On second thought it helps to have Armstong's thesis on Erlang as a bridge between them, but I can't find an easily downloadable link at the moment.)

EdwardCoffin on Nov 25, 2017 [-]

I also like the set of slides *Paranoid Programming - Techniques for Constructing Robust Software* [1] from Stratus Computer. They were discussed here earlier this year [2]

[1] http://ftp.stratus.com/vos/doc/papers/RobustProgramming.ps

[2] https://news.ycombinator.com/item?id=13678251

nickpsecurity on Nov 25, 2017 [-]

Joe Armstrong's thesis from Wayback Machine

https://web.archive.org/web/20151104200534/https://www.sics....

Guidelines | FAQ | Support | API | Security | Lists | Bookmarklet | Legal | Apply to YC | Contact

Search: