

Software Architecture is Hard

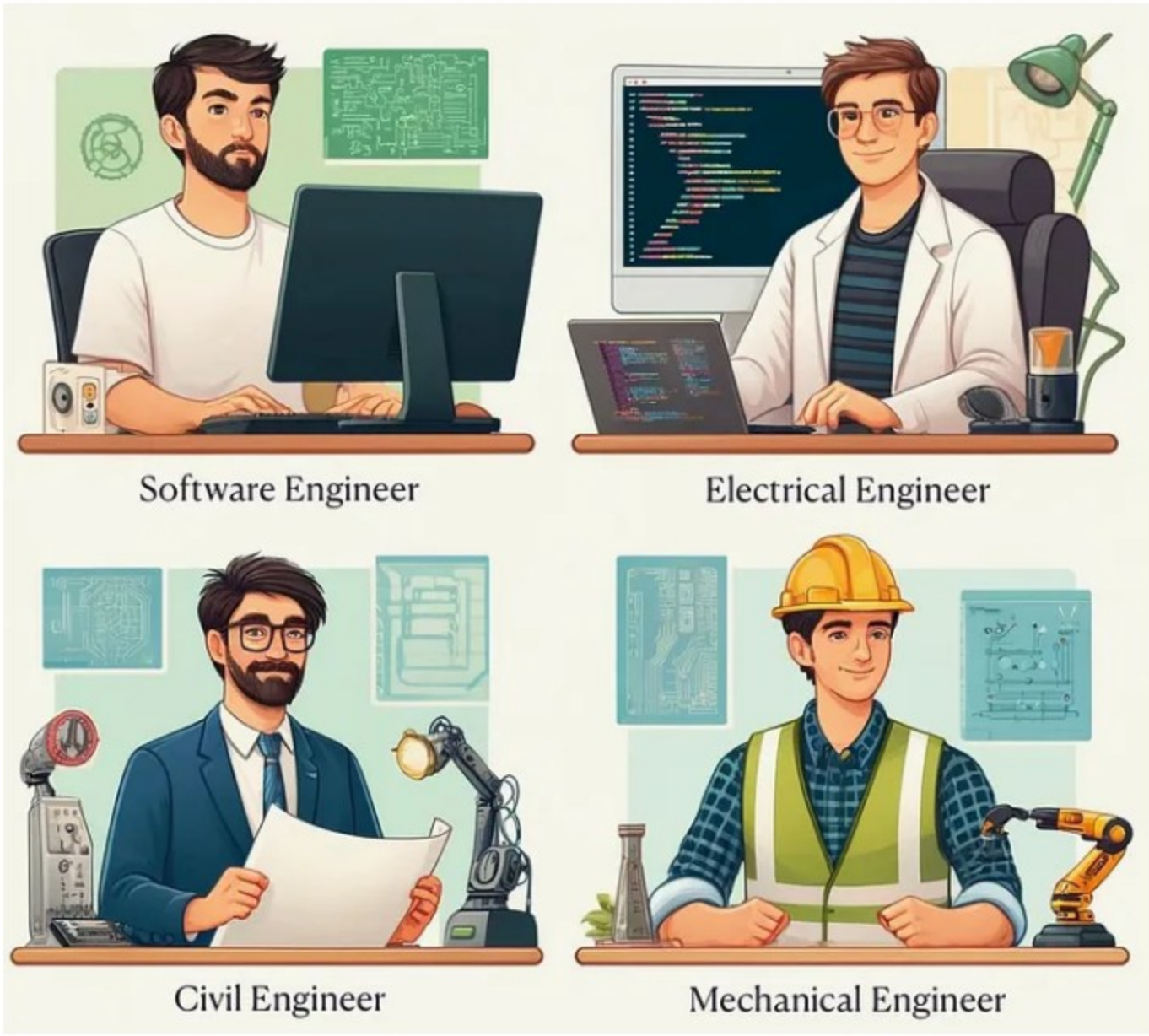
Oz Anani · Follow
7 min read · Apr 29, 2024

1.3K 34

I live in one of the suburbs of Tel-Aviv. One day, I woke up to insane drilling sounds. **It started!** the construction work to build the purple line of the light rail. “*How bad is it going to be?*” I thought to myself...
A few weeks in, sporadically waking up at 5am depending on when the construction work starts, my wife told me that it's time for us to move out. Desperately trying to build my case for staying at the apartment (I love the area!), I tried to find the construction plans, assuming that the drilling part won't take so long.

When I viewed the construction plans, as a software engineer, I was pretty amazed. They had detailed, concrete plans for the next 3 years. They separated it into 3 phases, with exact measurements and road structures for each phase. They had a solid timeline. And the most interesting part — **I was able to easily read it**, and it was clear to me, although I have 0 knowledge in civil engineering.

In fact, this expands to other types of engineering/architecture as well. Think about diagrams of house architecture, electrical engineering, or mechanical engineering — there are consistent ways to write these so all other engineers in the field can read.
But this is not the case for software engineering. While it's easy to create architecture diagrams in software (using boxes and arrows), reading and understanding them might not be straightforward.



Source: DALL-E
Prompt: Illustration of 4 types of engineers: software engineer, electrical engineer, civil engineer and mechanical engineer.

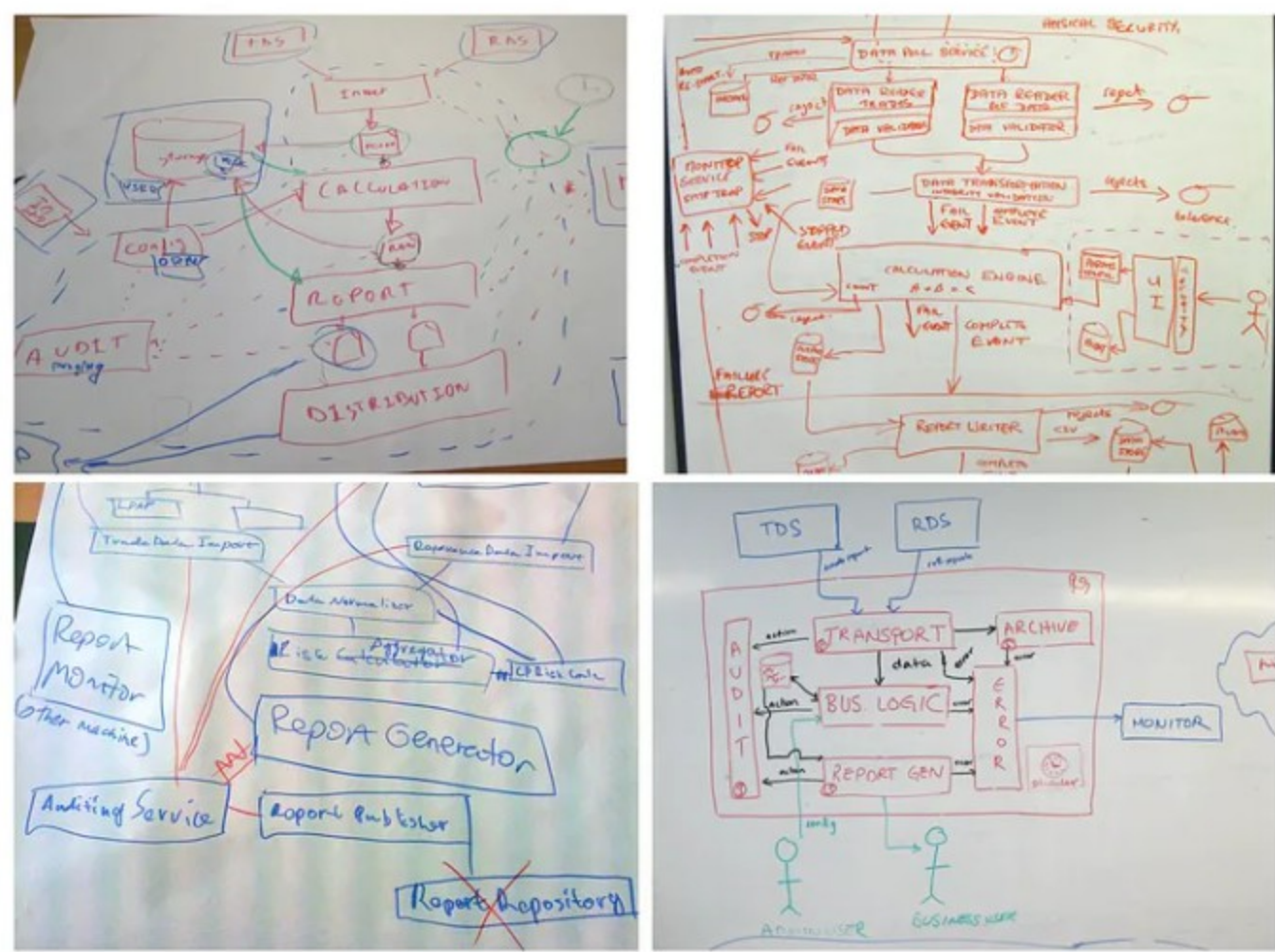
The Difference with Software Engineering

During my career as an engineer I've created, read and reviewed numerous software architecture diagrams and design docs, and conducted many system design interviews. I came to realize that **people describe software very differently**.

Differently, means:

- Different level of abstractions
- Different level of details
- Different illustration/diagramming techniques
- Different focus on non-functional aspects such as reliability, security, privacy and scale

It's common to find such **communication gaps even in the same team**, not to mention different teams, orgs or companies. These gaps manifest, for instance, in architecture diagrams that are not self-explanatory. They either come with significant wording (think 10-page design doc) or require significant time spent on explaining them.



Whiteboard architecture diagrams. Source: c4model.com

These communication gaps have even more impact due to a recent trend in the software engineering world.

Decentralizing the Architect Role

*“Architecture as a team sport:
Architects **no longer work alone**, and architects can no longer think only about technical issues. The role of an architect varies greatly across the industry, and some companies have eliminated the title entirely...”*

InfoQ Software Architecture and Design Trends Report — April 2023

In the past, most companies and organizations **implemented a top-down approach for software design** and architecture — employing an architect for each big enough group, that acts as the centralized authority for the product/system design. It was common for engineers to execute coding tasks based on given architecture, specification and APIs.

The world has shifted to a more decentralized approach, **where engineers at all levels now participate in designing and architecting**, rather than just coding. What varies between the different levels and skillset of engineers is the scope of their work — for instance, junior engineers could design a specific component, while more senior engineers can design a system, or a multi-system environment.

Basically, this is a cultural and mindset shift from the **top-down, centralized design process** to a **bottom-up, decentralized** and “owned by everyone” process.

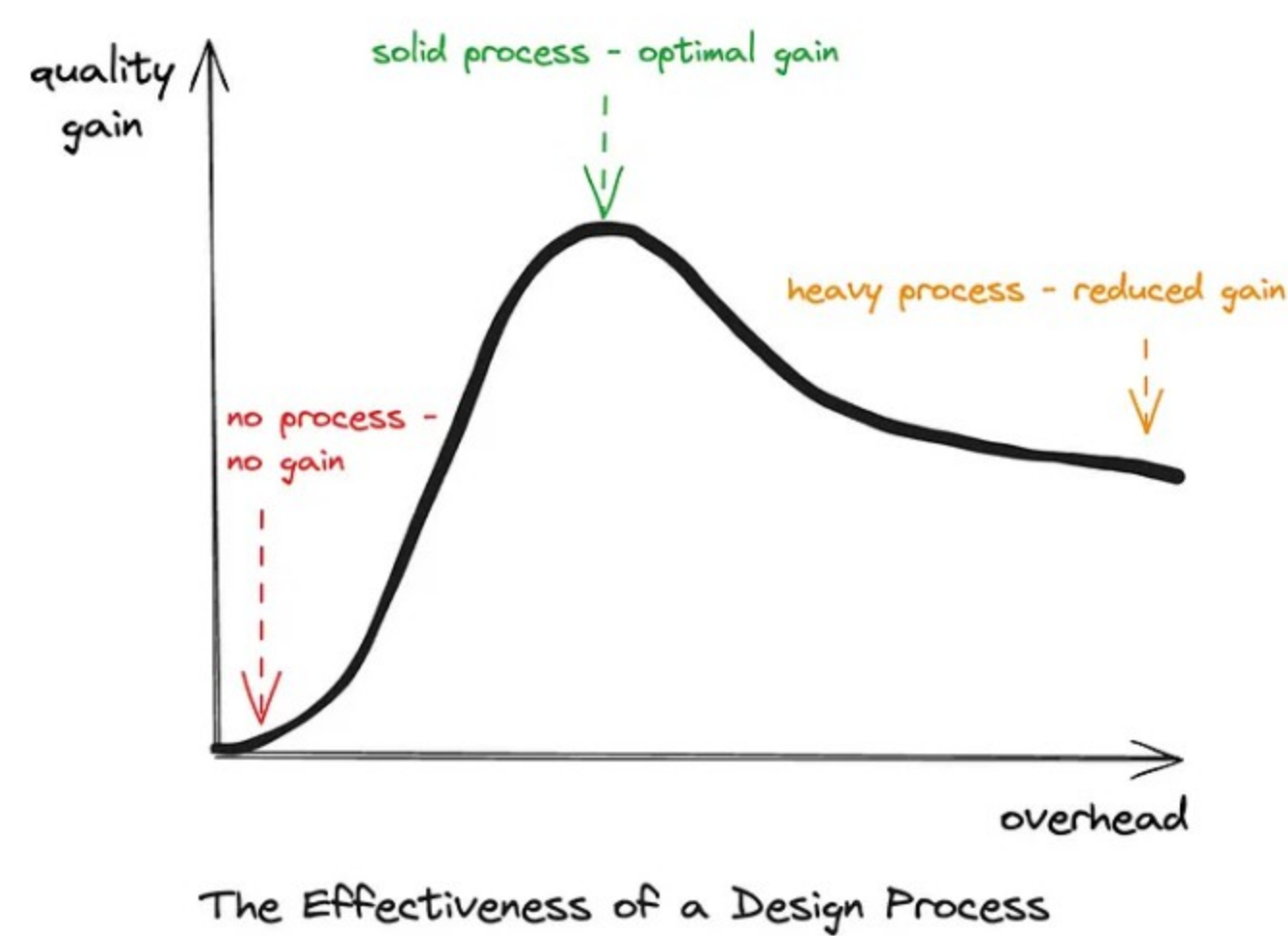
The Cost

This change **comes with many advantages**: empowering and growing engineers of all levels, increased sense of ownership and engagement, eliminating the single point of failure, and better spread of knowledge across teams.

However, it also comes with a cost:

- **Quality**: an architect is usually qualified as an experienced engineer with mileage in building and maintaining software systems. Spreading this work to everyone in the team means there is no further qualification aside of being part of the team. In order to keep an high quality bar, teams need to invest in training, education and mentoring, build a good design process and incorporate an open feedback culture.
- **Overhead**: having a solid design process in the org helps to increase the quality bar, but introduces a significant overhead. Usually such processes involve creating a design document, system diagrams, flow charts, and include a feedback process that can require multiple iterations and passes on the design.

Implementing a healthy design process in an organization is challenging. There is a tension between the overhead it introduces and the quality gain achieved by the process. If the process is too light, or does not exist at all, there is no quality gain. If the process is too heavy, it can become burdensome for engineers, who may not fully adhere to it or might even seek ways to circumvent it — ultimately leading to reduced efficiency. The sweet spot is something in between, as always.



There is a tension between the overhead a design process introduces and the achieved quality gain. If there is no process at all, there is no gain. If the process is too heavy, engineers will bypass it, and the gain will decrease. The sweet spot is something in between.

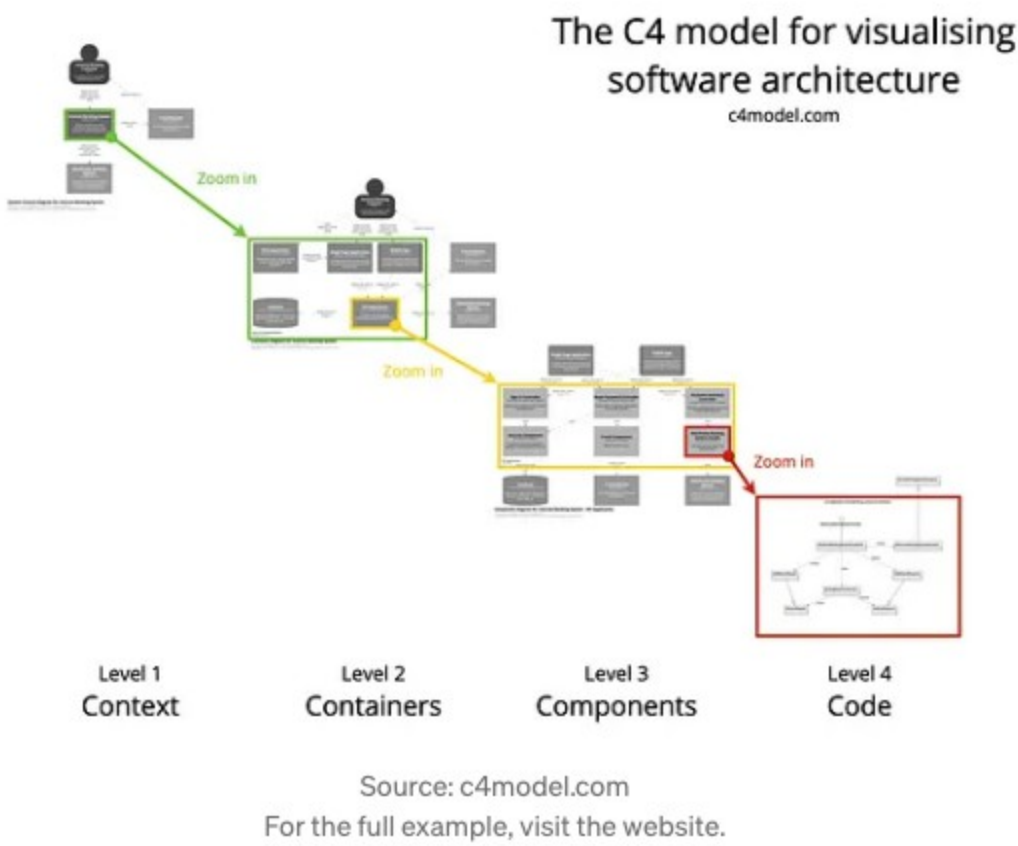
A Practical Takeaway: The C4 Model

There’s much to discuss on how a healthy design process should look like, but one practical concept I want to call out is the **C4 model**.

Visualizations are key part in software design. As mentioned above, people describe software very differently, especially when it comes to software diagrams.

Diagramming → Modeling

The C4 model aims to evolve the **short-lived, freestyle diagramming** techniques into a **long-lasting, canonical, and standardized modeling** technique. You can think of it as a set of rules and conventions on how software architecture should be visualized and described (similar to the standards that exist in other engineering fields).



The C4 model consists of 4 layers of abstractions (hence the name):

- 1. **System Context:** the highest level of abstraction. This layer contains internal systems, external systems, users (if there are multiple types of users, they should be specified) and the relationships between the 3 of them.
- 2. **Container:** once you have your systems in place, you can zoom into a specific system. A system is composed of one or more containers. A container **represents an application or a data store**, such as: backend service, frontend application, mobile application, relational DB, key-value store, message bus, object storage, etc.

An easy way to think about it, at least from a backend perspective, it to think about separate processes/executables. **Each executable maps to its own container.**

If you’ve had a system design interview before, it usually focuses on the container layer.

3. **Component:** next, a container can be **broken into several components, usually by functionality and areas of responsibility**. For instance, if we take a backend payment service, it could include components such as: payment executor, 3rd party client, recovery mechanism, payment recorder, notification sender, etc.

It’s ok that the components of a container will interact with other containers from layer 2 — for instance, the recovery mechanism could submit a message to the message bus to retry a failed payment in a certain delay.

4. **Code:** this is the mapping of components to actual code classes/modules/etc. This layer of implementation details is **usually less interesting and is very hard to maintain**, and in general not recommended for most use cases.

However, when writing code, it can be helpful to think “*In which component my code fits?*”; if the answer is not clear, either you’re creating a new component, or you’re violating the existing boundaries of components.

. . .

A few notes on the C4 model:

- **Container and Component are the critical layers.** Putting the essence of these layers clearly — your design choices on the Container layer impact scalability, reliability, deployment and evolvability. Your design choices on the Component layer impact developer velocity, efficiency, maintainability and quality.
- **The C4 model adds depth to software architecture diagrams.** Think of it as an online map — you start with the high level view, and can zoom in to places of interest. In case you got too much into the details, you can zoom out again and find yourself in the overarching architecture.
- **The C4 model is a collaborative, long-term model.** Instead of each engineer creating their own ad-hoc diagrams, the C4 model can be shared and worked on by multiple engineers or teams, expanding as the org and the product grow.
- **There are plenty of tools that support the C4 model.** Nowadays, even common enterprise diagramming software feature a C4 template.

Top highlight

Conclusion

Software design is a crucial part of the software development lifecycle, and has significant impact on the end result of the product or system, execution velocity, maintainability and evolvability.

With the recent trends of decentralizing the architect role and embracing “*architecture as a team sport*”, the structure of software teams is changing and organizations need to put careful thought on how their design process should look like and find the right balance between introducing overhead to maintaining a high quality bar.

Oz Anani

Another interesting question is how GenAI will affect software development and software design in the long-term. We already see changes in engineers’ day to day with the usage of AI tooling, co-pilots and agents.

How does the design process in your organization look like? do you feel it’s well-structured and impactful? or is there a place for improvement? would love to hear your thoughts via the comments below.

Software Architecture

Software Development

Software Engineering

Software Design

Programming



1.3K



34



Written by Oz Anani

306 Followers · 3 Following

Software Engineer at Meta | Creator of Swark - Automatic Architecture Diagrams from Code <https://github.com/swark-io/swark> | <https://linkedin.com/in/oz-anani>

Follow

Responses (34)



What are your thoughts?

Respond



Kornelije Petak
Jul 7, 2024



I think part of the issue is that this is a case of mixing apples and oranges. City infrastructure, buildings, roads and similar are designable on a different level than software, because those are created with all the requirements known in advance... [more](#)



29



2 replies

[Reply](#)



brett hoffman
May 10, 2024



I like C4 and the open sessions that explain it. In a world where every developer wants to be an architect — finally we are realising we all ARE! C4 is a great model to learn to diagram better together.

Was hoping to see your reflection on why our... [more](#)



33



1 reply

[Reply](#)



Milos Zivkovic
May 10, 2024



On my end C4 was good at start but slowly fades out of existence once the project matures. Unless there's someone who can update these diagrams when new requirements come in. Otherwise doesn't work as expected.



22



2 replies

[Reply](#)

See all responses

More from Oz Anani



Oz Anani

Introducing Swark: Automatic Architecture Diagrams from Code

Software architecture is hard to get right. Misaligned teams, unclear diagrams, and time-consuming processes often result i...



Jan 5



473



10



See all from Oz Anani

Recommended from Medium



In Javarevisited by Dylan Smith

Interview: How to Check Whether a Username Exists Among One...

My articles are open to everyone; non-member readers can read the full article by...



Aug 18, 2024



3.7K



49



Brian Jenney

3 Lessons from the Smartest Developers I've Worked With

I have a confession.

Oct 11, 2024



9.3K



167



Lists



General Coding Knowledge

20 stories · 1910 saves



Stories to Help You Grow as a Software Developer

19 stories · 1594 saves



Coding & Development


11 stories · 1005 saves



Leadership

62 stories · 517 saves




 Ali Zeynalli

How to Become a Strong Software Architect

Roadmap of a Software Architect

★ Jan 3 🖱 964 💬 18 



 In Level Up Coding by Lorenz Hofmann-Wellenhof

I Will Reject Your Pull Request If You Violate These Design...

4 Principles for Clean Code That Won't Get Your PR Rejected

★ Jan 29 🖱 660 💬 15 



 In Code Like A Girl by Nidhi Jain 🇮🇳

7 Productivity Hacks I Stole From a Principal Software Engineer

Golden tips and tricks that can make you unstoppable

★ Oct 15, 2024 🖱 5.6K 💬 119 



 Vinod Pal

How I Review Code As a Senior Developer For Better Results

I have been doing code reviews for quite some time and have become better at it...

★ Jan 26 🖱 164 💬 11 

See more recommendations