Thoughts on Software Architecture #3

Is UML finally dead? Generating API documentation, pragmatic software architecture, collection of free resources for developers

Kai Niklas Following
Jan 8 · 4 min read

💭 Thought of the Week

The recent Podcast with Martin Fowler caught my attention. He talked about his second edition of "Refactoring" book. And he talked about how he presents the refactoring patterns in his second edition. The first edition was mainly focused around Java samples with a tendency on class- and object-oriented patterns, but refactoring is not limited to that. Two major changes were made:

1) As it seems that today you can reach more people with JavaScript a switch has been made to give **code examples in JavaScript instead of Java**.

2) But an even more interesting change from an architects perspective is, that **instead of UML diagrams, almost exclusively code examples** are used to describe the refactoring patterns.

If even pattern books are not using UML anymore, I ask myself: Might UML be dead? If I look back the past 15 years **I have rarely used or have seen proper usage of UML in software development**. When scribbling a concept, pattern or architecture mainly two symbols are used: **Boxes and lines**.

Maybe some industries are still using it, but I don't know them? But even within model driven engineering (MDE): There are better things than standard UML to describe and generate code, i.e., domain specific languages (DSL) which usually deviate heavily from "standard" UML.

If we argue that UML is "just" a standardized language to help us describe or visualize software (architecture), then this looks differently. For example, I use standardized languages to describe business process with BPMN (Business Process Modelling Notation) or Archimate to describe or rationalize enterprise architecture. Very useful and helpful. But it's officially not UML.

Recently I stumbled across the C4 model to visualize software architecture. It uses different abstractions and looks similar to UML, but is not UML. It only uses UML (class) diagrams on the lowest level (code).

Further, agile development principles propagate to use less or no big-up-front-design (BUFD). This further reduces the usage of UML, as this is probably one of the bigger use cases for UML.

Updating diagrams in this context is cumbersome and often not done. What can be done instead is to generate an as-is view of the currently implemented system, e.g., in form of a UML diagram. This will help to better understand how the system looks like. But is it required to use UML for that purpose? Not necessarily…

🛠️ Tool / Method of the Week

▶ Generating API documentation with Swagger

Keeping documentation up to date is often cumbersome. It often simply does not work. We are all human beings and sometimes forget to update, miss information, or make some mistakes while documenting.

What if we could simply generate documentation based on the code or model we use(d) to implement our application. This assures everything is up to date.

One prominent use case is API documentation. Users need to understand how the API is used. And if models are used to describe the data model of the API in the fist place (contract first), then code and documentation can be automatically generated. Specification languages such as swagger provide the tools for that.

Find out more here:

- Documenting Your Existing APIs: API Documentation Made Easy with OpenAPI & Swagger

🏞️ Top Resources

▶ free-for.dev

> Developers and Open Source authors now have a massive amount of services offering free tiers, but it can be hard to find them all in order to make informed decisions.

▶ Adidas DevOps Maturity Framework

> The framework defines maturity levels for each capability that teams can use to self-assess their current maturity and understand what needs to be done to go to the next level.
>
> As practitioners Lean practitioners our framework is in continuous evolution and the "Run" of today might be the "Walk" of tomorrow, helping the teams to aim for continuous improvement.

▶ Software Architecture is Overrated, Clear and Simple Design is Underrated

> We did not use UML, nor the 4+1 model, nor ADR, nor C4, nor dependency diagrams. We created plenty of diagrams, but none of them followed any strict rules. Just plain old boxes and arrows, similar this one describing information flow or this one outlining class structure and relationships between components.
>
> Second, there were no architects on the teams that owned the design.
>
> Third, we had practically no references to the common architecture patterns.

▶ A Pragmatic Approach to Software Architecture

> A good architecture basically should:
>
> *- Try to account for as many future changes as possible*
> *- Make these changes easier for developers, not harder*

🏆 Project updates

▶ Become a Better Software Architect
(20% discount until Dec 24th, 2019)

Black Friday all around? Also my book was on sale. If you missed it, you can get 20% discount on my ebook on leanpub until December 24th, 2019. If you prefer something physical, you can also get it as paperback on amazon.

Wait, what is this?

Thoughts on Software Architecture is a newsletter with thoughts, resources, and commentary to help software engineers and software architects to create better software. It's written by Kai Niklas, who is a principal technology consultant, software architect, Ph.D. and author. If you like this newsletter I would be very grateful if you share it with people you like. If you are a first time reader of this newsletter, you can subscribe here and join over 500 peers.

. . .

Originally published at https://kniklas.substack.com.

Programming · Software Development · Software Architecture · Uml · API

57 claps

WRITTEN BY
Kai Niklas Following
Software Architect | Software Engineer | Open-Minded | Paraglider | PhD | Innovation Principal Consultant | https://bettersoftwarearchitect.com