Squads as a Service

Aggelos Bellos

- Architecture Team @ Epignosis
- Co-Organiser of Devstaff.gr
- Blogging about architecture.
- Hooked on Socio-Technical Systems

Squads As a Service

Embark on a journey through innovative software development and team dynamics. We'll explore how principles like microservices, along with SOLID and WET design patterns, shape not just our code, but also our teams. Discover the power of team topologies and modular thinking in creating adaptable, human-centric software solutions. Get ready to transform your teams into efficient units geared for technological excellence.

In the beginning, there was nothing

It was just you typing

Then you got your friend

And another one

Soon enough, you had a team

The team becomes too large

So now you have multiple teams / squads

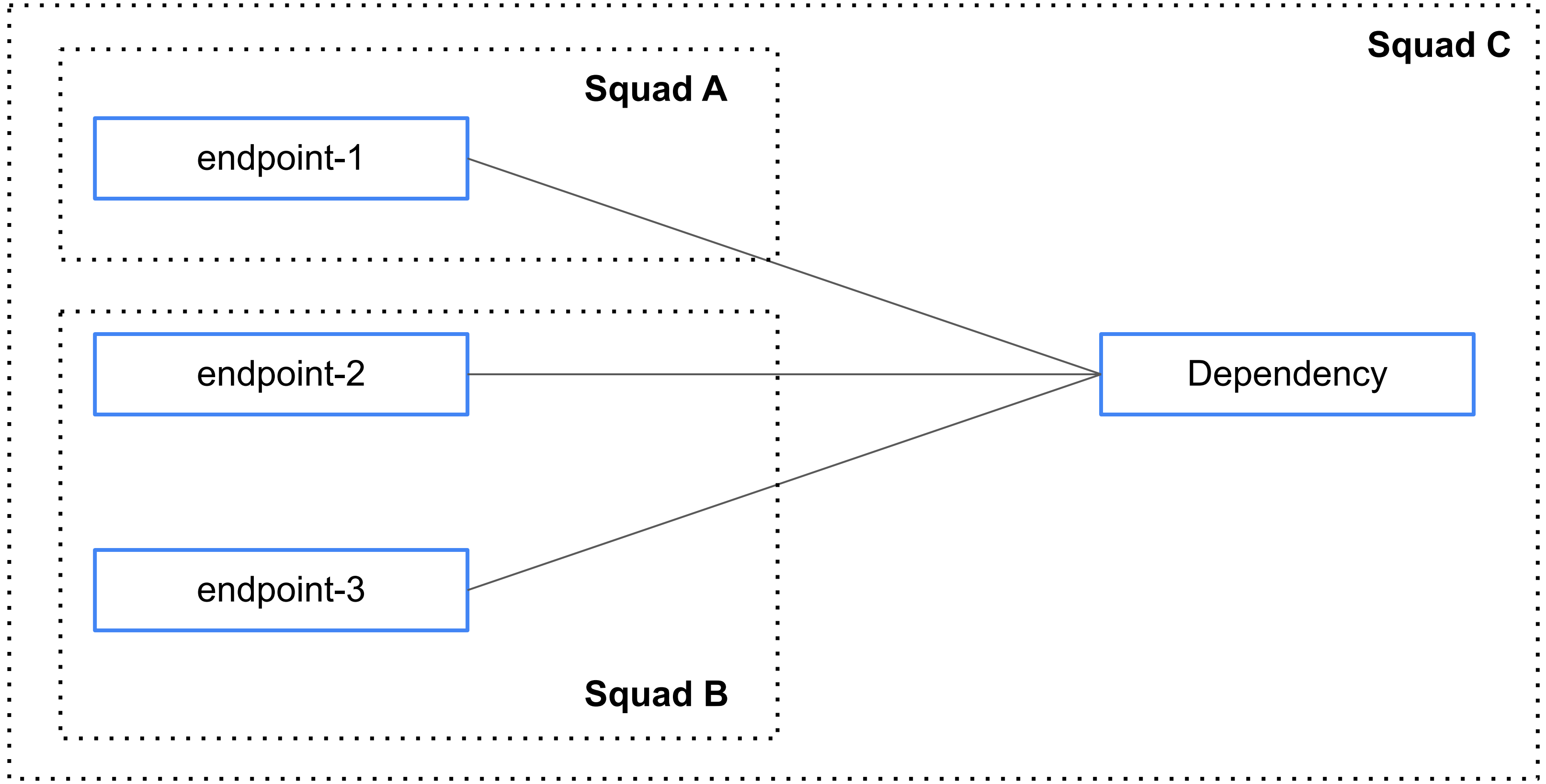
You did the good thing and splitted them based on the domain context

Management seems to be easier

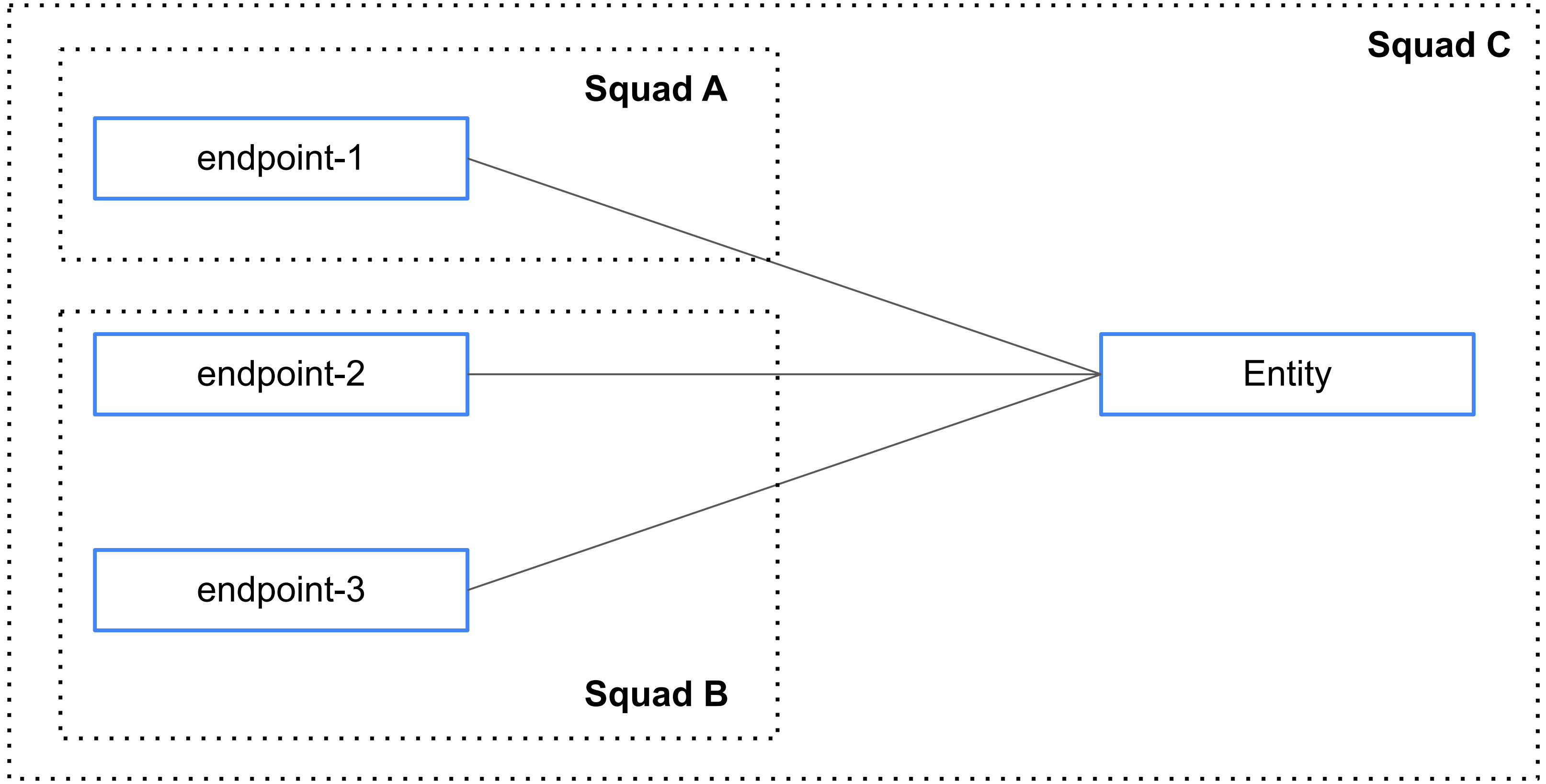
But the teams still have friction between them

Why is that?

Problem


Squad C
Squad A
endpoint-1
endpoint-2
Dependency
endpoint-3
Squad B

Problem


Squad C
Squad A
endpoint-1
endpoint-2
endpoint-3
Squad B
Entity

Shared dependencies

1. **Readability and Understandability**:
   Complex code or structures can be challenging to read and comprehend.

2. **Maintainability and Extensibility:**
   Complex code is more difficult to maintain and extend over time.

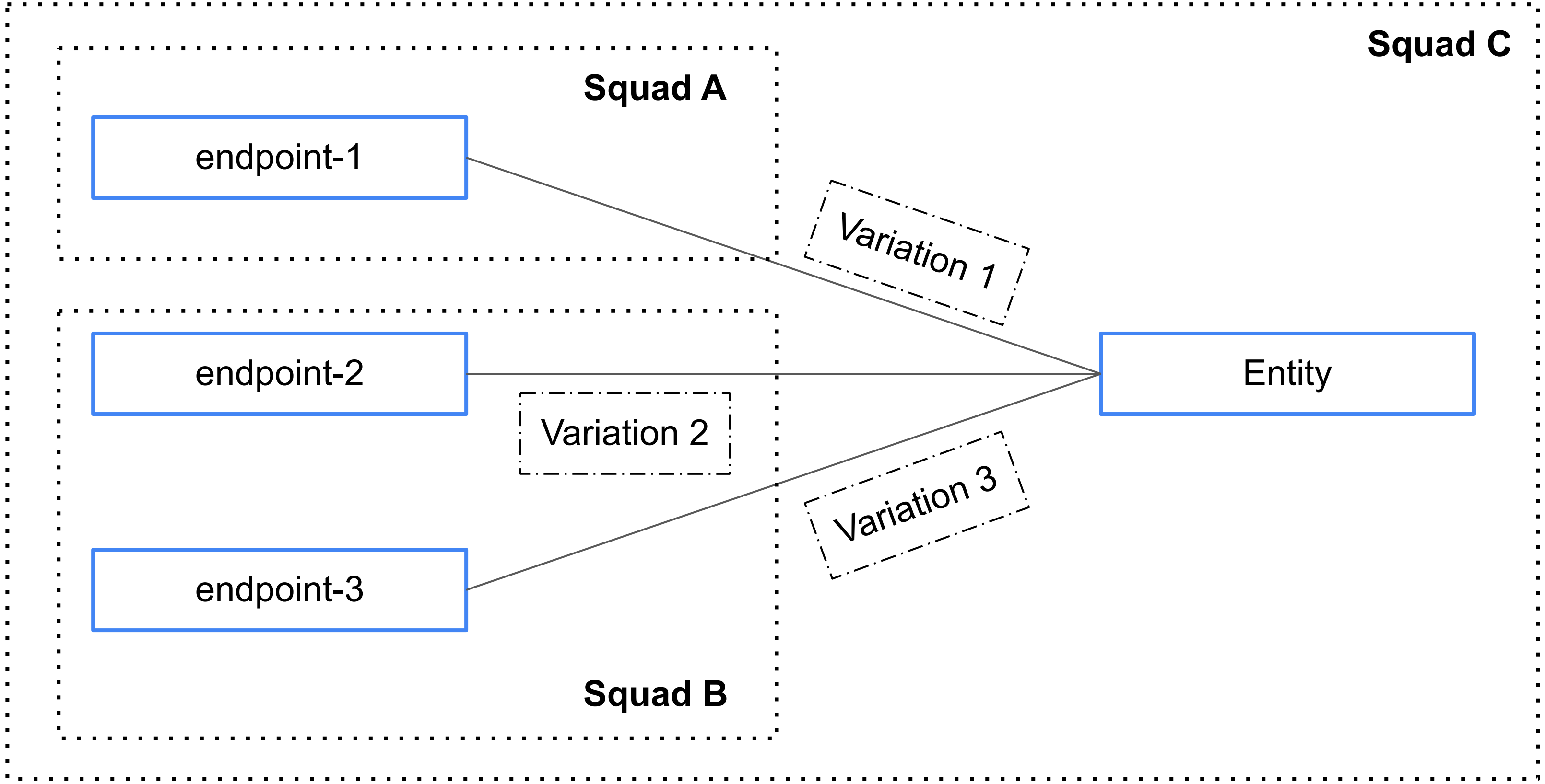
3. **Code Reusability:**
   Complex code tends to be less reusable.

These lead to:

- Performance and Efficiency issues
- Collaboration and Teamwork becomes more difficult
- ..and the worst of all…

Presentations by me

Problem


Squad C
Squad A
endpoint-1
Variation 1
endpoint-2
Variation 2
Entity
Variation 3
endpoint-3
Squad B

Traditionally, in monolithic architectures, we've leaned heavily on SOLID principles. These principles guide us towards creating more manageable, modular, and scalable codebases.

However, when we shift our focus to microservices, a different paradigm emerges - the WET (Write Everything Twice) approach.

The answer lies in the nature of microservices:

- **Independence and Autonomy:** Each squad can operate, update, and deploy their version of the entity independently, reducing coordination overhead and speeding up development cycles.

- **Resilience and Fault Isolation:** If one version of the entity fails or has a bug, it doesn’t necessarily bring down the others. This isolation is crucial in a distributed system where uptime and reliability are paramount.

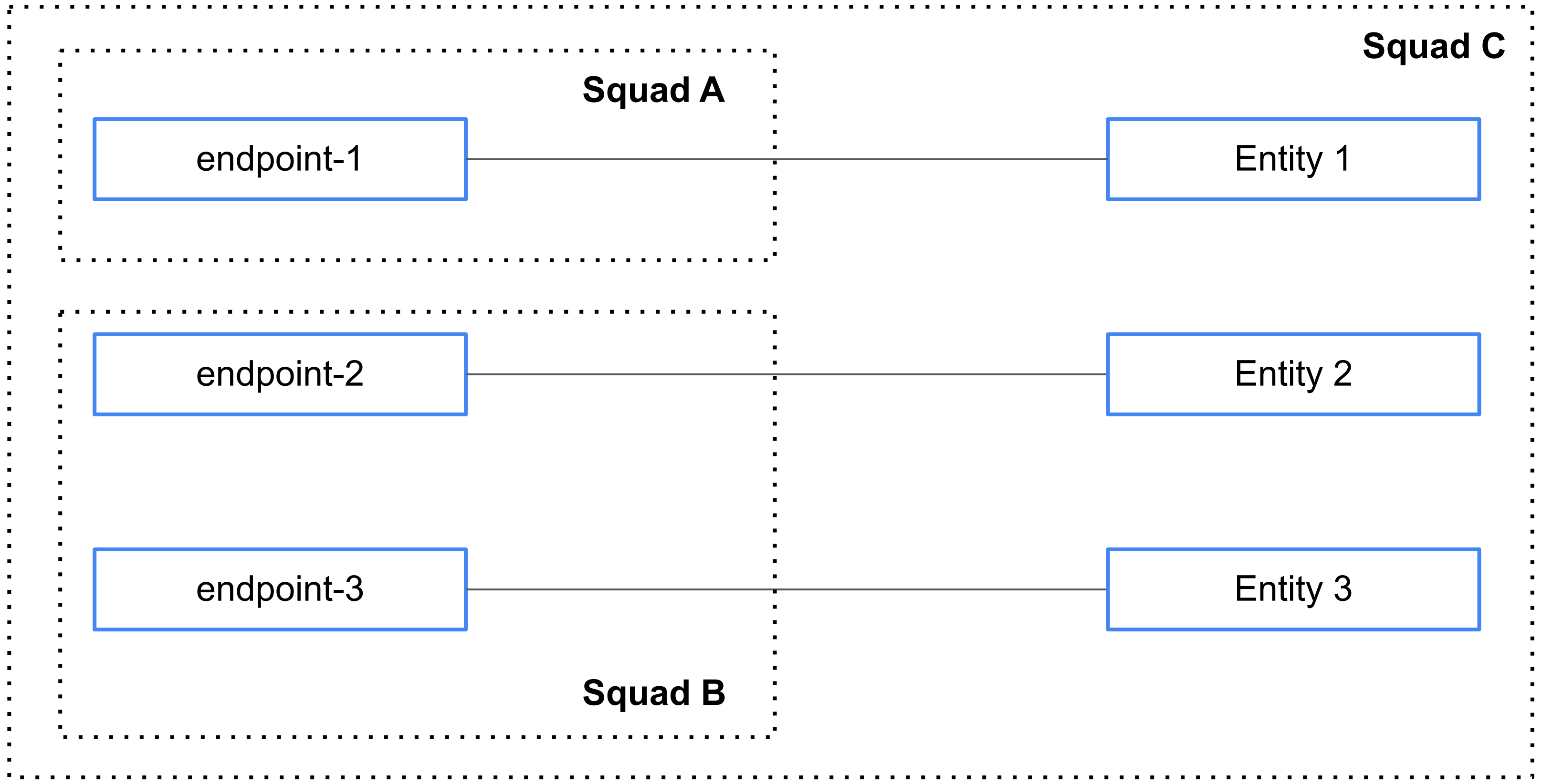
- **Flexibility and Customization:** Squads can tailor their version of the entity to their specific needs, rather than being forced to use a one-size-fits-all solution that might not be optimal.

Does WET applies only on microservices?

No

The key is to understand the context and needs of your project and team structure, and choose the approach that best aligns with your goals and challenges.

Solution



Squad C
Squad A
endpoint-1
Entity 1
endpoint-2
Entity 2
endpoint-3
Entity 3
Squad B

Single Responsibility Entity

1. Easy to comprehend.

2. Less conflicts between squads.

3. Easier to maintain.

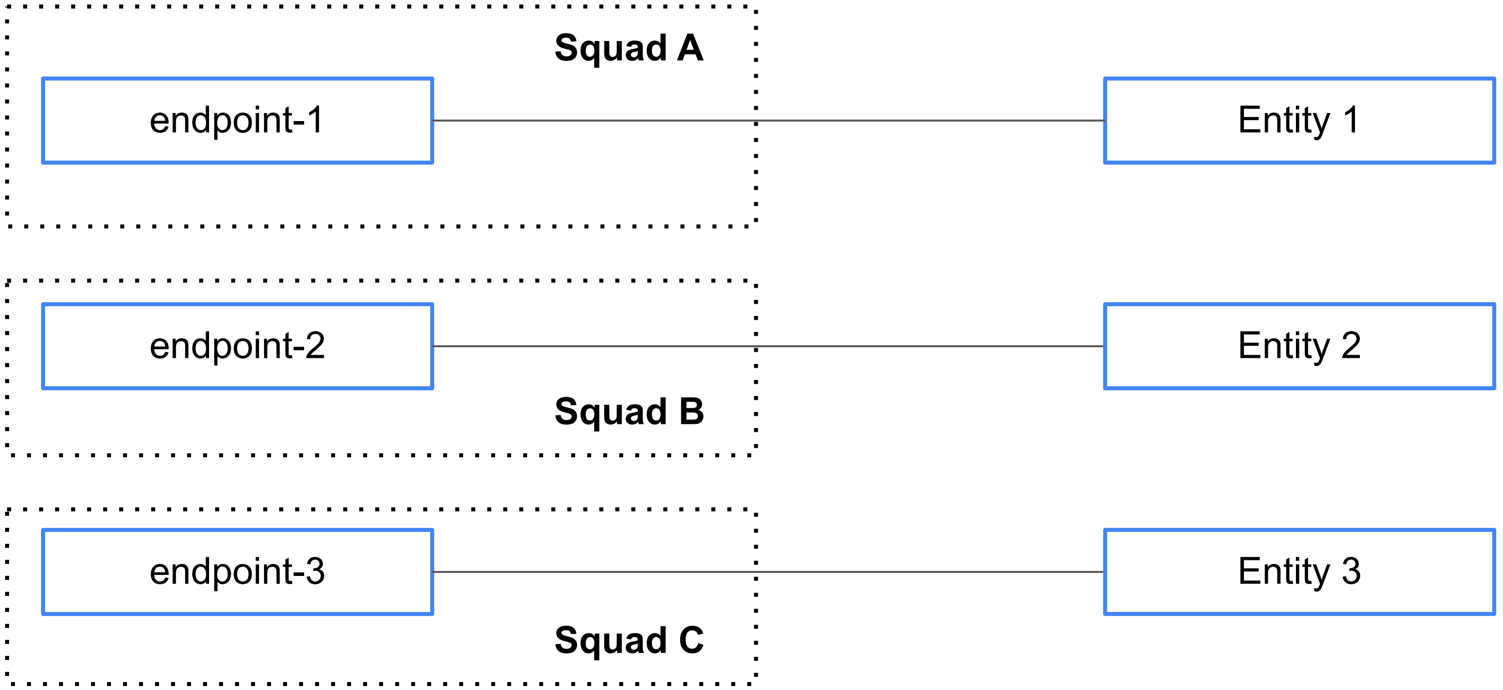
Growing our list of acronyms..

What is DDD (Domain-Driven Design)?

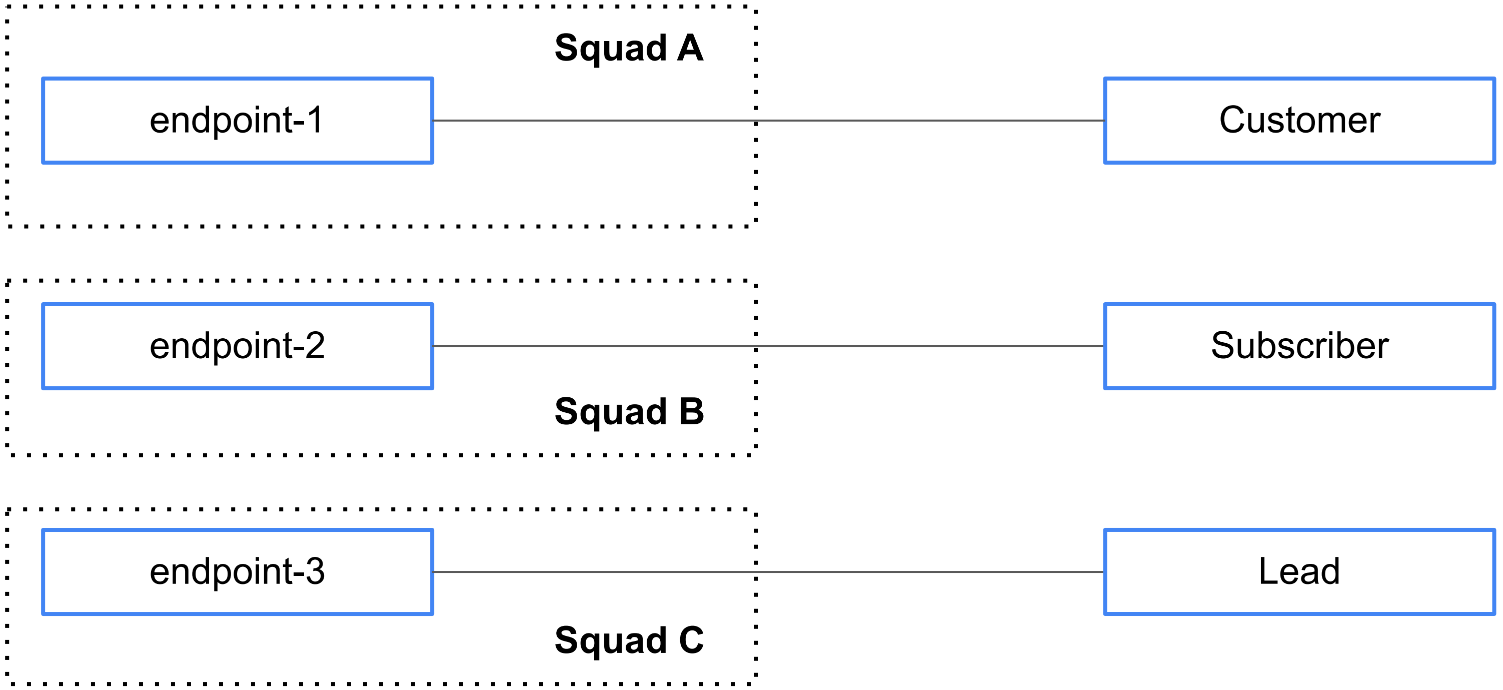
Domain-driven design (DDD) is a software development philosophy centered around the domain, or sphere of knowledge, of those that use it. The approach enables the development of software that is focused on the complex requirements of those that need it and doesn’t waste effort on anything unneeded. The clients of domain-driven design are often enterprise-level businesses.

Source: https://www.techtarget.com/whatis/definition/domain-driven-design

WET Approach



DDD Approach



We addressed several critical concerns:

Clarity and Relevance: Each squad works with an entity version that is directly relevant to their domain, making the code more intuitive and aligned with business logic.

Independence and Agility: Squads can develop, test, and deploy their services independently, leading to faster iteration and reduced dependencies.

Resilience: By isolating the domains, issues in one area are less likely to cascade into others, enhancing the overall stability of the system.

To sum up

Microservice

We have increased:

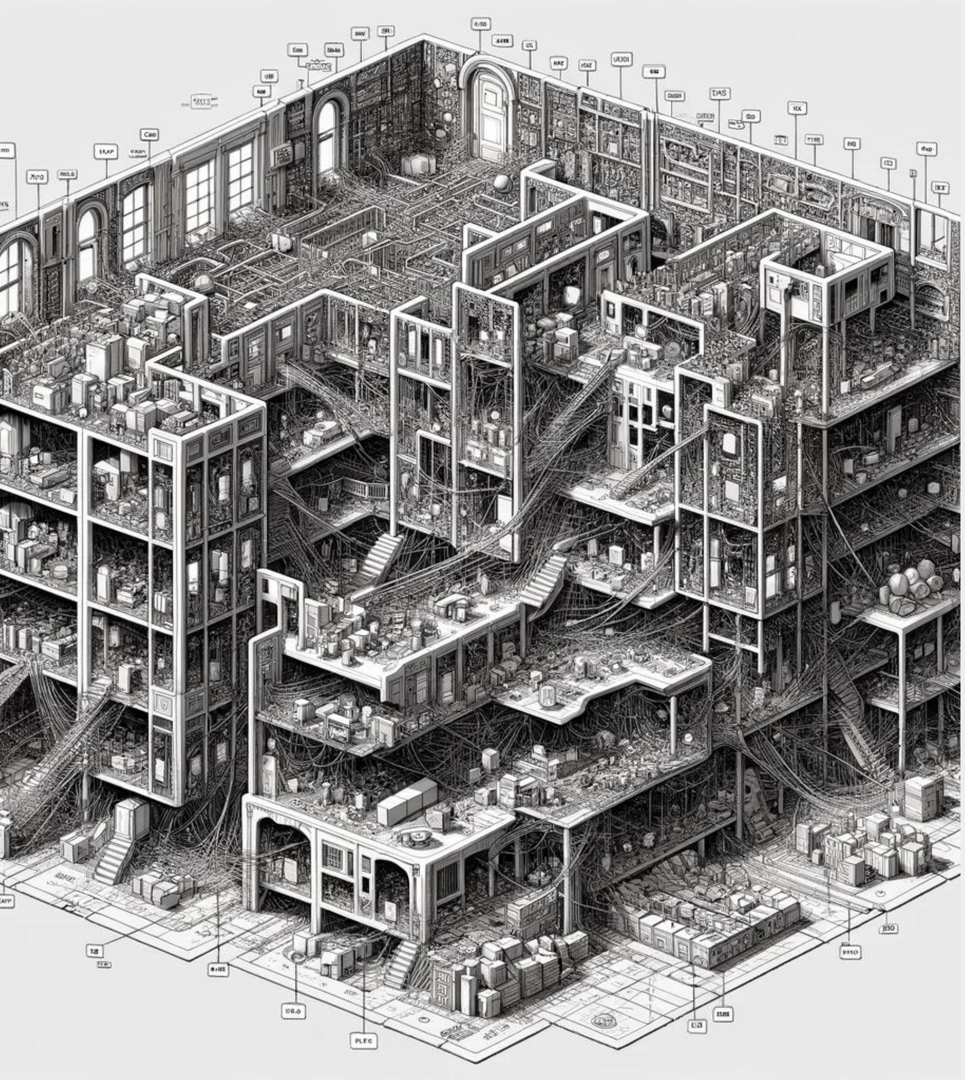
- Modularity
- Scalability
- Maintainability
- Fault Isolation

But why does this work?

Evaluating the Evolution of Software Arc
AI-Dominated Er
Brief Overview:
As AI technologies proliferate, the foundationa
architecture—crafted for a human-centric des
paradigm—are being challenged. This present
between classical software architecture and e
sparking a dialogue on the future role of archit
EPIGNOSIS

Does software architecture makes sense in a AI world?

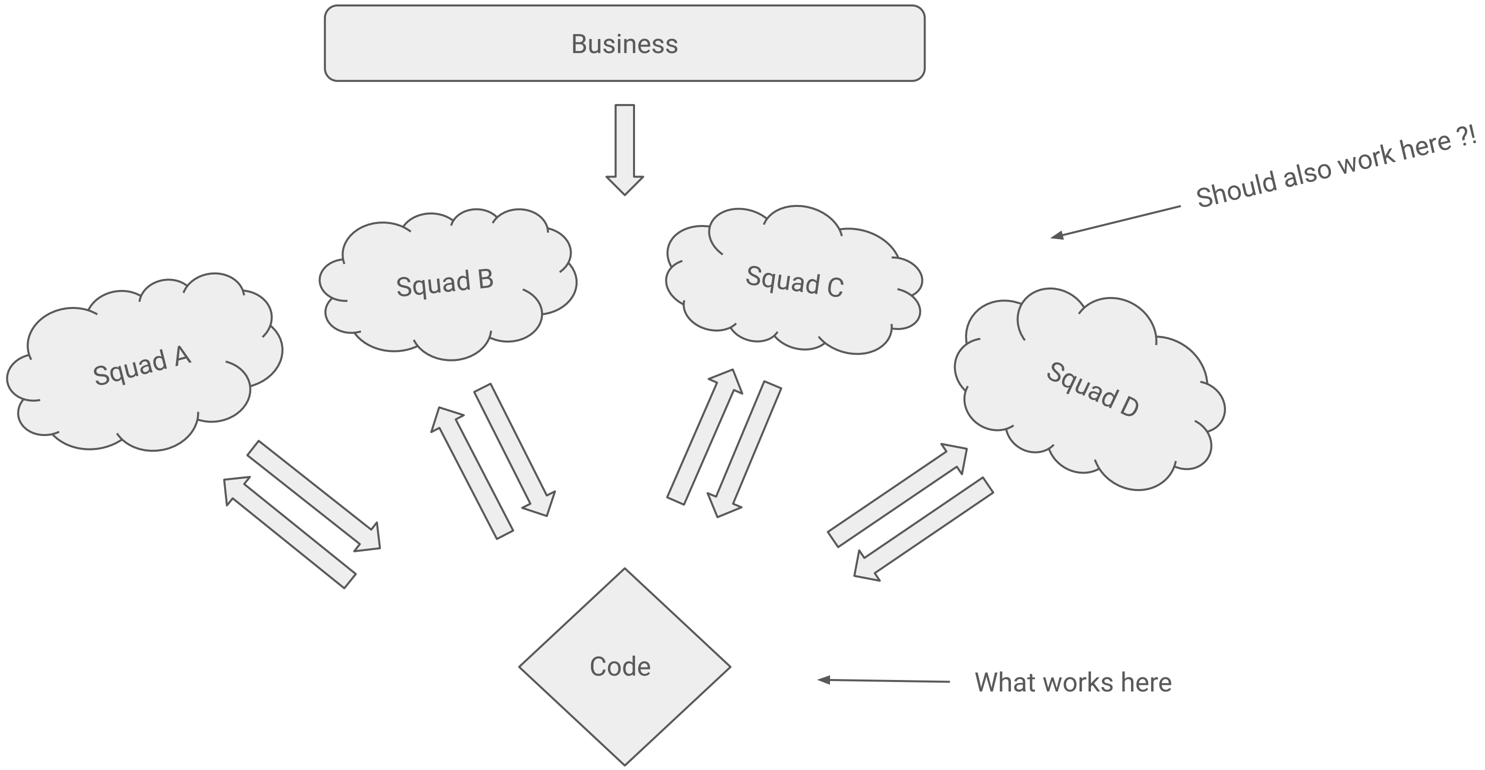
- Aggelos Bellos, Fosscomm 2023



[O]rganizations which design systems (in the broad sense used here) are constrained to produce designs which are copies of the communication structures of these organizations.

- Melvin E. Conway,
  How Do Committees Invent?

Ok, but now what?


Business
Squad A
Squad B
Squad C
Squad D
Should also work here ?!
Code
What works here

We can use already implemented frameworks and apply them to our squads.

What is a Dependency Graph?

A dependency graph is a visual representation of the dependencies between different elements of a system. In our context, these elements are the squads. The graph illustrates which squads are dependent on others and which squads act as dependencies

This visualization helps in identifying:

- **Inter-Squad Dependencies:** Clearly shows which squads rely on others for information, resources, or functionalities. It helps in understanding the flow of work and the potential bottlenecks.

- **Dependency Chains:** Long chains of dependencies can be risky as they increase the potential for delays and complications. Identifying these chains allows us to strategize on how to reduce or manage them more effectively.

- **Critical Paths:** Identifies squads that are critical for the workflow and which, if delayed, could impact the entire project.

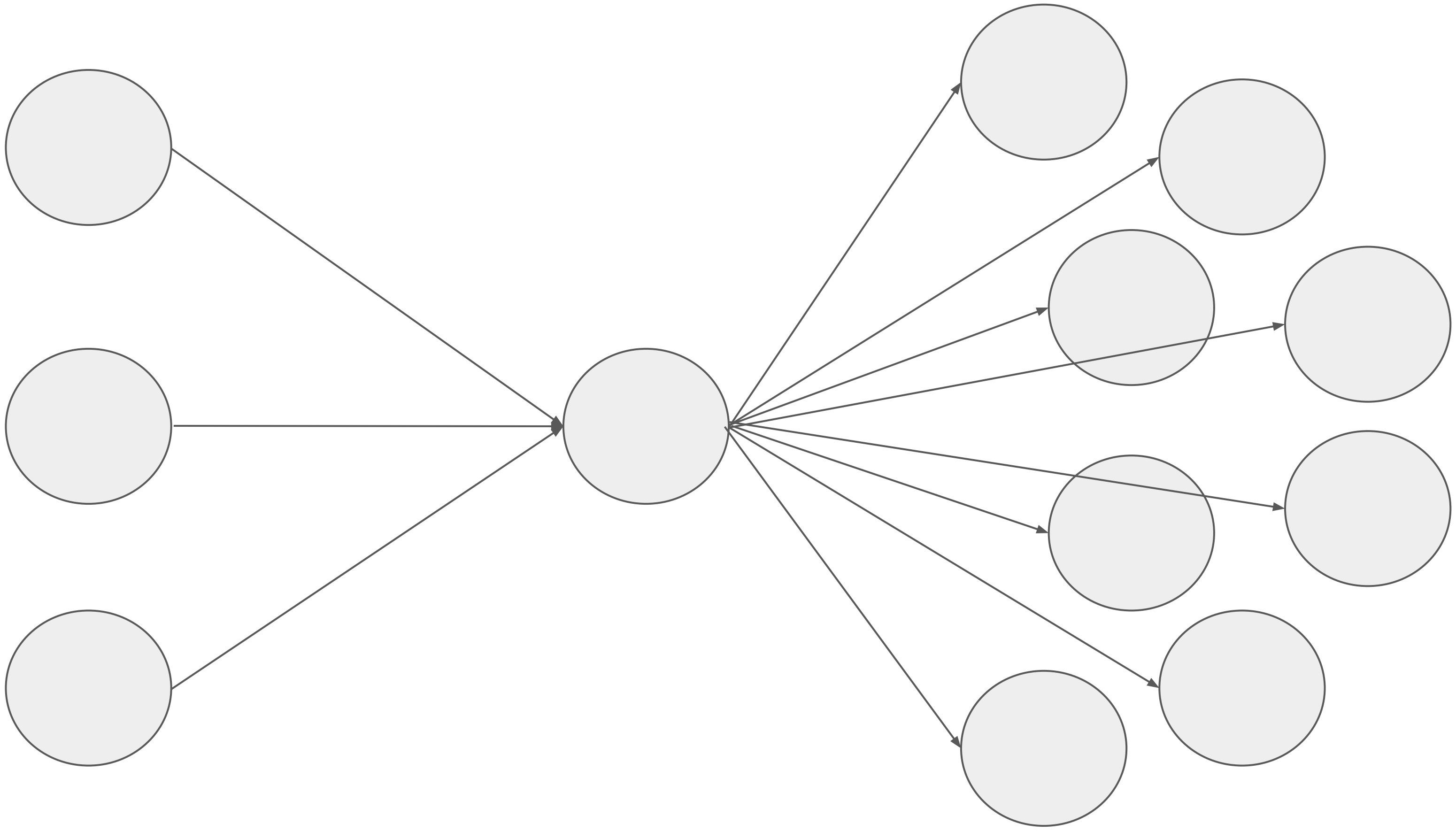
- **Areas for Optimization:** Helps in spotting areas where dependencies can be minimized or where communication can be improved.

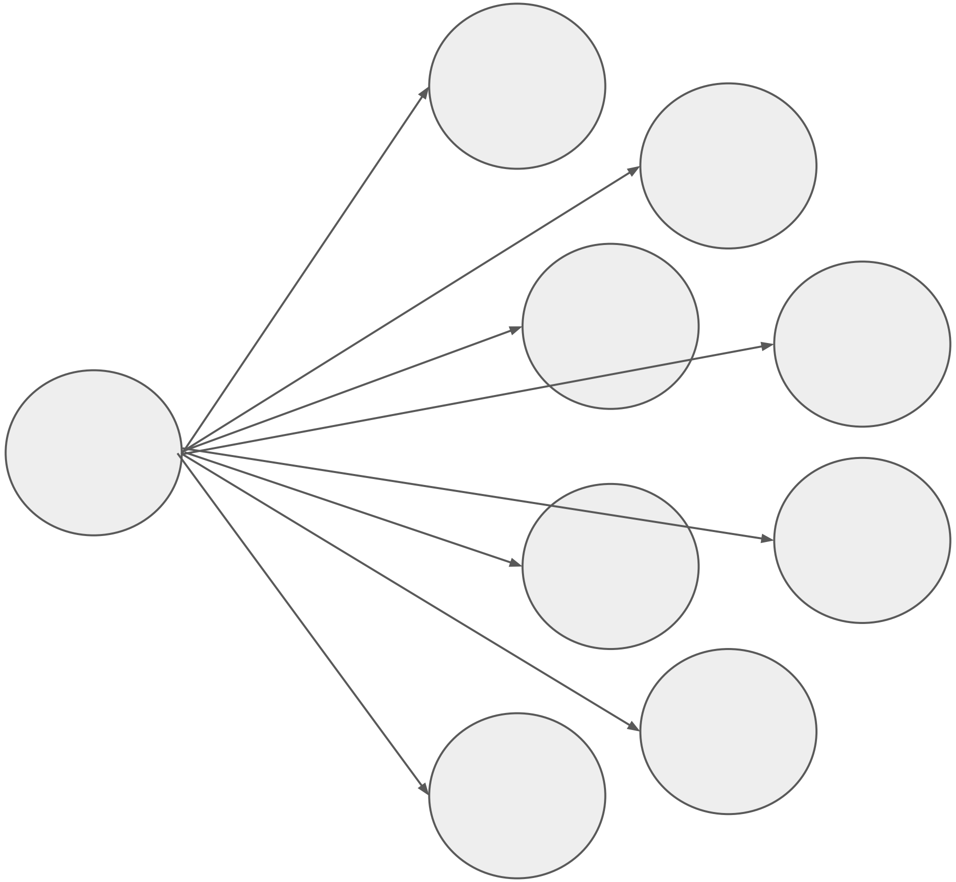
Creating and Utilizing Dependency Graphs

- **Data Collection:** The first step is to gather data about the interactions and dependencies between squads. This involves understanding the flow of work, the points of interaction, and the nature of dependencies.

- **Graph Construction:** Using this data, construct a graph where each node represents a squad, and the edges represent dependencies. Tools like Graphviz, Mermaid, or even specialized software like JIRA can be used for this purpose.

- **Analysis:** Once the graph is created, analyze it to identify key dependencies, potential bottlenecks, and areas where the workflow can be streamlined.



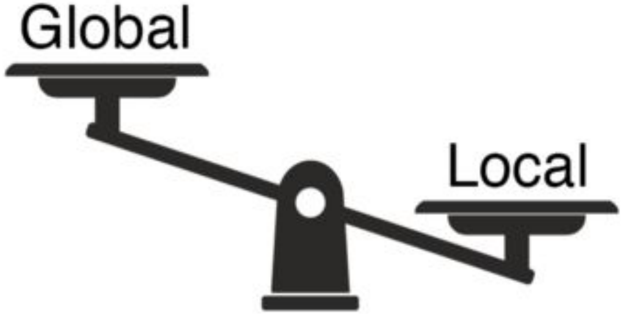


Iterative Updates

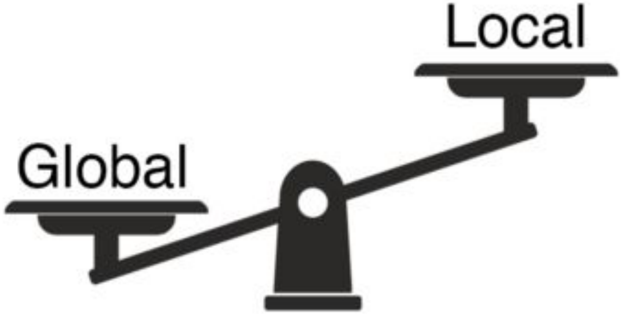
Dependency graphs are not static. They should be updated regularly to reflect the evolving nature of projects and teams.

Communication and Collaboration

Share the graph with all squads. It serves as a communication tool, helping everyone understand their place in the larger ecosystem and fostering a sense of collaboration.



Big Ball of Mud



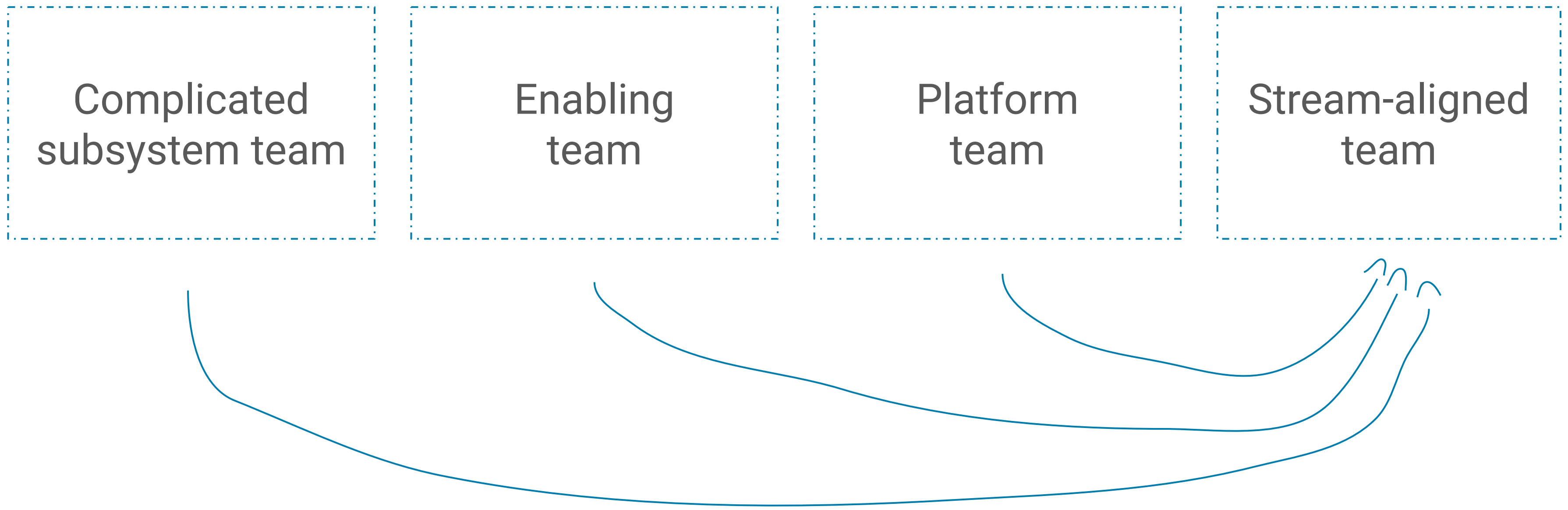
Distributed
Big Ball of Mud

Our design patterns and architectural choices are fundamentally about enhancing how we, as humans, interact with and develop software.

Communication and architecture mean nothing without a focus

Team Topologies

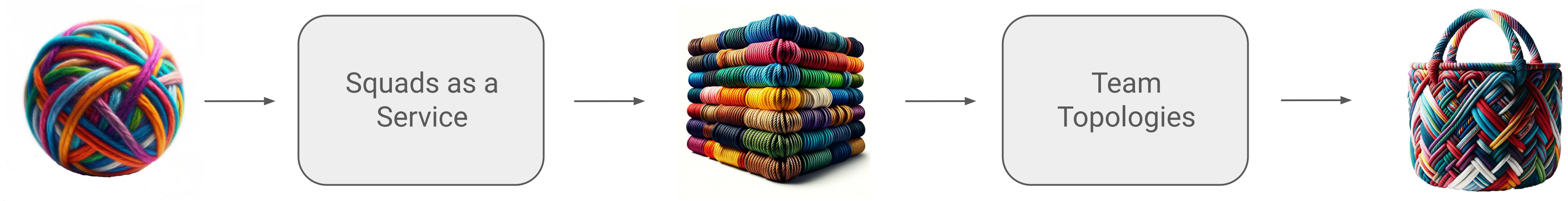
A concept developed by Matthew Skelton and Manuel Pais, provides a framework for organizing software development teams in a way that complements and enhances the architecture of the systems they build.

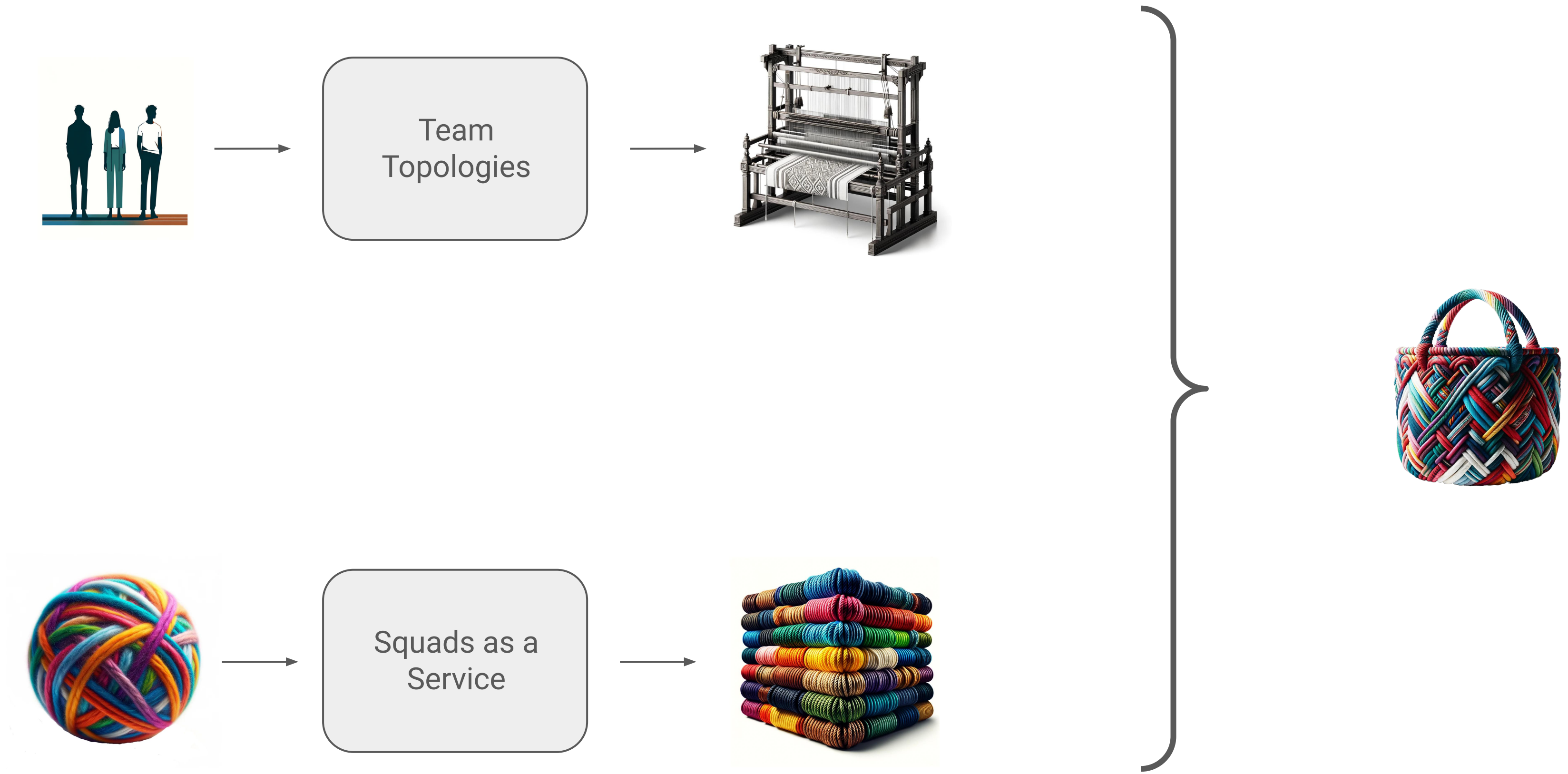

Complicated subsystem team
Enabling team
Platform team
Stream-aligned team

By aligning our team structures with these topologies, we not only reflect the modularity and resilience of our microservices but also enhance the human element of our work.

These team structures foster collaboration, innovation, and efficiency.

They ensure that our teams are organized in a way that maximizes their strengths and aligns closely with our architectural choices.


Squads as a Service
Team Topologies


Team Topologies
Squads as a Service

Do these answers still make sense?

Iterative Process

DDD - SOLID - WET

Answers the question: In What?

In **what** part of the code do I contribute?

Team Topologies

Answers the questions: How? Where?

How do I contribute to the flow and **where** do I exist?

Any Questions?

linkedin.com/in/aggelos-bellos/

aggelosbellos7@gmail.com

medium.com/@aggelosbellos