Antipattern: Horizontalism

Author(s): Andreas, Christian

this antipattern describes the division of applications into layers managed by individual teams, hindering the delivery of end-user value.

Description: What is it, and what are the bad consequences?

Horizontalism occurs when software applications are structured into layers, with each layer managed by a separate team. While this may seem organized, it leads to a significant drawback: no team can independently deliver end-user value. Instead, each team's output depends heavily on coordination with others, causing development to slow down.

The consequences of Horizontalism include:

1. Dependency Bottlenecks: Teams become overly reliant on each other, causing bottlenecks in development as they wait for dependencies to be resolved.
2. Communication Overhead: Constant communication between teams becomes necessary to coordinate efforts, leading to inefficiencies and delays.
3. Reduced Agility: The rigidity of interdependent layers makes it challenging to adapt quickly to changing requirements or market demands.
4. Difficulty in Debugging, Testing and Maintenance: With multiple teams responsible for different layers, diagnosing, writing tests and fixing issues becomes complex and time-consuming.
5. Feature Envy: One layer of the software implements the responsibilities or capabilities of another layer. E.g. business logic in frontend layers.

What are some examples?

- Dedicated frontend teams which only consume JSON from backend teams.
- Separate database teams, which implement stored procedures containing business logic.

Why does this happen?

Horizontalism often arises due to a misguided attempt at achieving architectural purity or division of labor. Teams may perceive layer-based separation as a logical means of organizing responsibilities, but fail to recognize the negative impact on agility and delivery speed.

Moreover, organizational structures that prioritize departmental silos or lack clear communication channels increase the problem. When teams work in isolation, without understanding the broader context or end-user needs, they unintentionally contribute to Horizontalism.

How can we avoid getting into the situation in the first place?

To prevent falling into the trap of Horizontalism, consider the following strategies:

1. Cross-functional Collaboration: Encourage collaboration among teams from the outset, fostering a shared understanding of objectives and user needs.
2. Domain-driven Architecture: Design applications with independent submodules, which can be developed and deployed independently, by a individual teams.
3. Clear Communication Channels: Establish clear communication channels and encourage open dialogue between teams to mitigate misunderstandings and streamline collaboration.

What are suggestions to get out of the situation if we ended up in it?

If Horizontalism has already taken root in your software architecture, consider the following steps to mitigate its effects:

1. Restructure Teams: Evaluate the possibility of restructuring teams previously responsible for single layers to cross-functional teams.
2. Refactor Architecture: Break down layers incrementally into smaller, better manageable vertical components. For example, into modules inside a modulith.