Antipattern: Under-Modularization

Author(s): Andreas, Theo, Christian, Felix

Software systems are divided into too few modules at run time (e.g. containers) or build time (e.g. Maven modules, packages). This modularization has disadvantages compared to a modularization into more modules. The modules are too big and therefore harder to understand, maintain, secure, deploy and scale horizontally. Due to missing boundaries in the code the risk of dependencies and data sharing across different functions increases.

What are some examples?

- Big Ball of Mud
- certain monolithic applications
- Two huge domain classes represent all legal forms for companies, unions and foundations

Why does this happen?

- Old project structures. E.g. certain COBOL applications.
- Uncontrolled software growth.
- Time pressure.
- Low budget.
- Organization issues: No clear vision for the overall modularization.
- A lack of architecture and technology knowledge/experience.
- A lack of domain knowledge.

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

- Create meaningful modules which emphasize important architectural boundaries. Cut your modules by bounded contexts. (Domain-driven Design)
- Do internal and external reviews regularly to get feedback to your modularization early.
- Split your project into more modules as soon as you benefit from it. For example, when:
    - Code is reused
    - Code is maintained by another team
    - The performance is insufficient at build/run time
    - Modules get too big/complex

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

- Split modules based on the reviews of the modularization.
- Rewrite or refactor the system or its parts.