Antipattern: Over-Modularization

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

Software projects are divided into several modules at run time (e.g. services, lambdas) or build time (e.g. Maven modules, packages). This modularization has disadvantages compared to a modularization into fewer modules. It adds *unnecessary* complexity. Therefore, the project becomes harder to understand and debug for developers. Over-Modularization could lead to higher compile times. Additionally, the performance at run time could be inferior compared to a modularization into fewer modules.

What are some examples?

- Cutting Maven modules or packages too small.
- Cutting microservices/nanoservices/lambdas too small.
- Temporal coupling of modules.
- Splitting a checkout system into too many services

Why does this happen?

- premature optimization:
  - Developers plan a modularization that will be ideal in the future.
  - The modules are supposed to provide a better build-performance/maintainability/testability in the future.
  - The modules could be reused in the future.
  - Developers are afraid that a further modularization at a later time will be difficult.
- Cargo-Culting: Developers modularize software because others do without understanding why.
- Developers made bad experiences with Under-Modularization.
- Developers lack domain knowledge.

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

- Try to have few meaningful modules which emphasize important architectural boundaries. Cut your modules by bounded contexts. (Domain-driven Design)
- Yagni: Don't split your project into more modules until you need to.
- Reevaluate the project's modularization periodically and refactor your software accordingly. Start early, later will be harder.

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

- Develop a target modularization based on the current problems and experience. Compare it to the current modularization.
- Refactor modules based on the evaluation of the modularization.
- Merge modules which are usually deployed/modified together.