ToiletPaper #118

jambit
WHERE INNOVATION WORKS

Architecture in software systems – How pattern languages increase readability and code comprehension

Author: Matthias Mair / Senior Software Architect / Business Division Automotive Bavaria

✗ Problem

It's a typical scenario – a development team stands in the coffee kitchen and criticizes the bad or completely missing architecture in the software system. This scenario occurs more often in maintenance and further development projects of older software systems. Age plays a decisive role in this case. The older a system is, the higher is the probability of finding a strongly eroded architecture [1].

But what does the development team see as architecture in this scenario and what is missing?

✓ Solution

The development team probably lacks a consistent and uniform use of *pattern languages* [2] in the existing software system, which would significantly increase readability and code understanding. Pattern languages go one step further than architectural styles (layers, microservices etc.) and provide structuring specifications for classes (or modules, packages etc.) on a hierarchically more detailed level. A pattern language defines a set of pattern elements and specifies rules for them. These rules determine which responsibilities the individual pattern elements have and how they may interact with each other (mandatory and prohibited rules) [3]. In other words: pattern languages define the use of design patterns and their uniform use throughout the software system.

It is never too late in a project to deal with pattern languages and use them in an individual context – consistently and uniformly throughout the code. Documentation of the pattern language is not only helpful but also recommended. It does not matter whether it is a maintenance or greenfield project. In this sense, pattern languages are a good tool to allow more time in the coffee kitchen for more in-depth discussions about cat videos and computer games.

→ Example

1) Chart of the business logic in the service layer

Pattern element	Mandatory and prohibition rules
Service interface	Interface in the package com.example.<productname>.service.api with the naming convention *Service
Service implementation	Class in package com.example.<productname>.service with the annotation @Service Naming convention *Service<xyz> (default: xyz = Impl)

2) Using Spring Web MVC in the project (excerpt)

Pattern element	Mandatory and prohibition rules
Model	Is created and filled by the pattern element "Controller"; contains no logic
View	Template language (e.g. Thymeleaf with templates (*.html) in the resources/templates directory) Has only access to the pattern element "Model" and not to "Controller"
Controller	Defines the endpoints using @RequestMapping annotations and their specializations The class has the annotation @Controller and as naming convention *Controller Controllers are stateless and contain no business logic

+ Further Aspects

- [1] M. Mair and S. Herold, „Towards Extensive Software Architecture Erosion Repairs" in Software Architecture. 7th European Conference, ECSA 2013, Montpellier, 2013.
- [2] Christopher Alexander, A Pattern Language. Towns, Buildings, Construction
- [3] C. Lilienthal, Langlebige Software-Architekturen: Technische Schulden analysieren, begrenzen und abbauen, dpunkt.verlag, 2015.