ToiletPaper #118

Architektur in Softwaresystemen: Wie Mustersprachen Lesbarkeit und Code-Verständnis erhöhen

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

✗ Problem

Wer kennt dieses typische Szenario unter Entwicklern nicht? Ein Entwicklungsteam steht in der Kaffeeküche zusammen und moniert die schlechte oder gänzlich fehlende Architektur im Softwaresystem. Dieses Szenario tritt verstärkt bei Wartungs- und Weiterentwicklungsprojekten von älteren Softwaresystemen auf. Das Alter spielt hierbei eine entscheidende Rolle. Je älter ein System ist, desto höher ist die Wahrscheinlichkeit, eine stark erodierte Architektur [1] vorzufinden.

Was versteht jedoch das Entwicklungsteam in diesem Szenario unter Architektur und was fehlt ihm?

✓ Lösung

Dem Entwicklungsteam fehlt wahrscheinlich eine konsistente und einheitliche Verwendung von *Mustersprachen* [2] im vorliegenden Softwaresystem, welche die Lesbarkeit und das Code-Verständnis deutlich erhöhen würden. Mustersprachen gehen dabei einen Schritt weiter als Architekturstile (Schichten, Microservices etc.) und geben auf einer hierarchisch detaillierteren Ebene Strukturierungsvorgaben für Klassen (bzw. Module, Packages etc.). Eine Mustersprache definiert jeweils ein Set von Musterelementen und gibt dafür Regeln an. Diese Regeln legen fest, welche Verantwortlichkeiten die einzelnen Musterelemente haben und wie diese miteinander interagieren dürfen (Gebots- und Verbotsregeln) [3]. In anderen Worten: Mustersprachen definieren den Einsatz von Entwurfsmustern und deren einheitliche Verwendung im *gesamten* Softwaresystem.

Im Projekt ist es nie zu spät, sich mit Mustersprachen zu beschäftigen und diese im individuellen Kontext einzusetzen - und zwar konsistent und einheitlich im gesamten Code. Eine Dokumentation der Mustersprache ist nicht nur hilfreich, sondern auch empfehlenswert. Es spielt hierbei keine Rolle, ob es sich um ein Wartungs- oder Greenfield-Projekt handelt. In diesem Sinne, sind Mustersprachen ein gutes Hilfsmittel, damit in der Kaffeeküche mehr Zeit für tiefgreifendere Gespräche zu Katzenvideos und Computerspielen bleibt.

➔ Beispiel

1) Abbildung der Geschäftslogik in der Service-Schicht

Musterelement	Gebots- und Verbotsregeln
Service-Interface	Interface im Package com.example.<produktname>.service.api mit der Namenskonvention *Service
Service-Implementierung	Klasse im Package com.example.<produktname>.service mit der Annotation @Service Namenskonvention *Service<xyz> (Standard: xyz = Impl)

2) Verwendung des Spring Web MVC im Projekt (Auszug)

Musterelement	Gebots- und Verbotsregeln
Model	Wird vom Musterelement Controller erstellt und befüllt; enthält keine Logik
View	Template-Sprache (z. B. Thymeleaf mit Templates (*.html) im Verzeichnis resources/templates) Hat nur Zugriff auf das Musterelement Model und nicht auf den Controller
Controller	Definiert die Endpoints über @RequestMapping-Annotationen und deren Spezialisierungen Die Klasse hat die Annotation @Controller und als Namenskonvention *Controller Controller sind zustandslos und enthalten keine Geschäftslogik

+ Weiterführende Aspekte

- [1] M. Mair und 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.