

Clean Architecture

Praxisbuch

für saubere Software-Architektur und wartbaren Code

DAS INHALTS- VERZEICHNIS

» Hier geht's
direkt
zum Buch

Inhaltsverzeichnis

	Vorwort	9
	Mitwirkende	11
	Einleitung	15
1	Wartbarkeit	19
1.1	Was bedeutet Wartbarkeit überhaupt?.....	19
1.2	Wartbarkeit führt zu Funktionalität.....	20
1.3	Wartbarkeit erzeugt Entwicklerfreude.....	23
1.4	Wartbarkeit unterstützt die Entscheidungsfindung.....	25
1.5	Die Wartbarkeit bewahren.....	26
2	Was ist so falsch an Schichten?	29
2.1	Sie fördern ein datenbankgetriebenes Design.....	30
2.2	Sie sind anfällig für Abkürzungen.....	32
2.3	Sie werden immer schwieriger zu testen.....	33
2.4	Sie verbergen die Use Cases.....	34
2.5	Sie erschweren das parallele Arbeiten.....	36
2.6	Wie hilft Ihnen dies, wartbare Software zu entwickeln?.....	37
3	Abhängigkeiten umkehren	39
3.1	Das Single-Responsibility-Prinzip.....	39
3.2	Eine Geschichte über Nebenwirkungen.....	41
3.3	Das Dependency-Inversion-Prinzip.....	41
3.4	Clean Architecture.....	43
3.5	Hexagonale Architektur.....	45
3.6	Wie hilft Ihnen dies, wartbare Software zu entwickeln?.....	48
4	Code organisieren	49
4.1	Nach Schichten organisieren.....	49
4.2	Nach Features organisieren.....	50
4.3	Eine Architektur-explicite Paketstruktur.....	52
4.4	Die Rolle der Dependency Injection.....	54
4.5	Wie hilft Ihnen dies, wartbare Software zu entwickeln?.....	56

5	Einen Use Case implementieren	57
5.1	Das Domänenmodell implementieren.	57
5.2	Ein Use Case im Überblick.	59
5.3	Die Eingabe validieren.	61
5.4	Die Macht von Konstruktoren.	64
5.5	Unterschiedliche Eingabemodelle für unterschiedliche Use Cases	66
5.6	Business-Regeln validieren	67
5.7	Reiches versus anämisches Domänenmodell	69
5.8	Unterschiedliche Ausgabemodelle für unterschiedliche Use Cases	70
5.9	Was ist mit »read only« Use Cases?	71
5.10	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	72
6	Einen Web-Adapter implementieren	73
6.1	Dependency Inversion.	73
6.2	Die Verantwortlichkeiten eines Web-Adapters	75
6.3	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	80
7	Einen Persistenz-Adapter implementieren	81
7.1	Dependency Inversion.	81
7.2	Verantwortlichkeiten eines Persistenz-Adapters.	82
7.3	Portschnittstellen aufteilen	83
7.4	Persistenz-Adapter aufteilen	85
7.5	Ein Beispiel mit Spring Data JPA	87
7.6	Was ist mit Datenbanktransaktionen?	92
7.7	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	93
8	Architekturelemente testen	95
8.1	Die Testpyramide.	95
8.2	Testen eines Domänen-Entitys mit Unit-Tests	97
8.3	Testen eines Use Case mit Unit-Tests	98
8.4	Testen eines Web-Adapters mit Integrationstests.	100
8.5	Testen eines Persistenz-Adapters mit Integrationstests	102
8.6	Testen der Hauptpfade mit Systemtests	104
8.7	Wie viel Testen ist genug?	109
8.8	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	110
9	Mapping über Grenzen hinweg	113
9.1	Die »No Mapping«-Strategie.	113
9.2	Die »Zwei-Wege«-Mapping-Strategie.	115

9.3	Die »vollständige« Mapping-Strategie	117
9.4	Die »Ein-Weg«-Mapping-Strategie	118
9.5	Wann wird welche Mapping-Strategie benutzt?	119
9.6	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	121
10	Die Anwendung zusammensetzen	123
10.1	Warum ist das Zusammensetzen wichtig?	123
10.2	Konfiguration mit einfachem Code	125
10.3	Konfiguration mit Classpath-Scanning von Spring	126
10.4	Konfiguration mit Java Config von Spring	129
10.5	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	131
11	Bewusst Abkürzungen nehmen	133
11.1	Wieso Abkürzungen wie eingeschlagene Fenster sind	133
11.2	Die Verantwortung, sauber zu beginnen.	134
11.3	Modelle zwischen Use Cases teilen	135
11.4	Domänen-Entities als Ein- oder Ausgabemodell benutzen	137
11.5	Eingangsports überspringen.	138
11.6	Services überspringen.	139
11.7	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	140
12	Architekturgrenzen erzwingen.	143
12.1	Grenzen und Abhängigkeiten	143
12.2	Modifier für die Sichtbarkeit	144
12.3	Fitnessfunktion nach dem Kompilieren	146
12.4	Build-Artefakte.	149
12.5	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	153
13	Mehrere Bounded Contexts verwalten	155
13.1	Ein Hexagon pro Bounded Context?	156
13.2	Entkoppelte Bounded Contexts	158
13.3	Angemessen gekoppelte Bounded Contexts	160
13.4	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	162
14	Ein komponentenbasierter Ansatz für die Softwarearchitektur.	165
14.1	Modularität durch Komponenten.	166
14.2	Fallstudie – eine »Check Engine«-Komponente erstellen	169
14.3	Komponentengrenzen validieren.	171
14.4	Wie hilft Ihnen dies, wartbare Software zu entwickeln?	173

15	Sich für einen Architekturstil entscheiden	175
15.1	Beginnen Sie einfach	175
15.2	Entwickeln Sie die Domäne	176
15.3	Vertrauen Sie auf Ihre Erfahrung	177
15.4	Es kommt drauf an	177
	Stichwortverzeichnis	179