

Inhaltsverzeichnis

Einleitung	11
Teil I Grundlagen	27
I Objektorientierte Programmierung – diesmal richtig	29
I.1 Die prozedurale Programmierung	29
I.2 Die objektorientierte Programmierung	30
I.2.1 Abstraktion	32
I.2.2 Datenkapselung	32
I.2.3 Vererbung	32
I.2.4 Polymorphie	33
I.3 Vorteile der objektorientierten Programmierung	36
I.4 Die mathematische Theorie der abstrakten Datentypen	37
I.5 Die objektorientierte Modellierung	37
I.6 UML	39
I.7 Das Problem mit dem Umlernen	40
I.8 Das Problem mit der Zweckentfremdung von Programmiersprachen	41
2 Automatisierte Tests	43
2.1 Fehler in Programmen	43
2.2 Methoden zum Aufspüren von Fehlern	44
2.3 Werkzeuge zur Laufzeitüberprüfung	45
2.4 SUnit und die testgetriebene Softwareentwicklung	46
2.5 JUnit	47
2.5.1 Methoden zur Überprüfung von Testergebnissen	49
2.5.2 Methoden für Vor- und Nacharbeiten	50
2.5.3 Test-Suiten	50
2.6 Fallbeispiel: Maximum-Bestimmung	53
2.7 Testimplementierung (Test Double)	57
2.7.1 Teststrunk (Test Stub)	60

2.7.2	Testatruppe (Mock Object)	60
2.7.3	Fake-Objekt (Fake Object)	60
2.7.4	Dummy-Objekt (Dummy Object)	60
2.8	Arten von automatisierten Tests	61
2.8.1	Entwicklertests	61
2.8.2	Externe Tests	61
2.8.3	Abnahmetests (Acceptance Tests)	61
2.8.4	Integrationstests (Integration Tests)	62
2.8.5	Lasttests (Load Tests, Performance Tests)	62
2.8.6	Andere Arten der Kategorisierung	62
2.9	Eigenschaften guter Tests	63
2.9.1	Sorgfältig	63
2.9.2	(Voll-)Automatisch	63
2.9.3	Performant	64
2.9.4	Vollständig	64
2.9.5	Wiederholbar	65
2.9.6	Unabhängig	65
2.9.7	Stabil	65
3	Entwicklungsprinzipien der objektorientierten Programmierung	69
3.1	Interface-Aufteilungsprinzip (Interface Segregation Principle)	69
3.2	Liskov-Substitutionsprinzip (Liskov Substitution Principle)	71
3.3	Abhängigkeits-Inversionsprinzip (Dependency-Inversion Principle)	73
3.4	Einzelzuständigkeitsprinzip (Single-Responsibility Principle)	75
3.5	Erweiterungsprinzip (Open-Closed Principle)	78
4	Entwurfsmuster der objektorientierten Programmierung	81
4.1	Abstrakte Fabrik (Abstract Factory)	82
4.2	Schablonenmethode (Template Method)	82
4.3	Wertobjekte (Value Object)	83
4.4	Null-Objekt (Null Object)	85
4.5	Stellvertreter (Proxy)	89
4.6	Adapter (Adaptor)	91
4.7	Beobachter (Observer)	91
4.8	Fassade (Facade)	95
4.9	Kommando (Command, Transaction, Operation)	99

4.I0	Strategie (Strategy)	103
4.II	Weitere Entwurfsmuster	108
4.I2	Wie man Entwurfsmuster nicht verwendet	108
5	Refactoring	111
5.1	Refactoring in unterschiedlichen Größenordnungen	113
5.2	Umbenennen von Bezeichnern	114
5.3	Methode extrahieren (Extract Method)	114
5.4	Methode auflösen (Inline Method)	116
5.5	Methode verschieben (Move Method)	116
5.6	Methode hochziehen (Pull-Up Method)	118
5.7	Interface extrahieren (Extract Interface)	120
5.8	Klasse extrahieren (Extract Class)	120
5.9	Ein Schritt nach dem anderen	121
5.I0	Refactorings skizzieren	122
6	Fehlerbehandlung	123
6.1	Die Anfänge	123
6.2	Ausnahmebehandlung	126
6.3	Der dreiphasige Prozess	127
6.3.1	Fehlerauslösung	128
6.3.2	Fehlerweiterleitung	130
6.3.3	Gruppieren von Fehlern	131
6.3.4	Fehlerprotokollierung und Kompensation	131
6.4	Ein Beispiel	132
Teil II Weiterentwicklung bestehender Systeme ohne vorhandene Tests		135
7	Bestehende Systeme mit Tests erweitern	137
7.1	Irgendwo muss man anfangen	137
7.2	Fachlogik unabhängig vom Bestandssystem entwickeln	138
7.3	Bestandssystem gegen Testimplementierung entwickeln	149
7.4	Beispiel mit komplexerer Fassade	151
7.5	Schritt für Schritt zu besserem Code	152
8	Abschotten neuer Funktionalitäten durch implizite Tests	155
8.1	Währungsobjekt mit unzureichender Parametervalidierung	155
8.2	Verbesserte Kommunikation durch implizite Tests	163

Teil III Bestehende Systeme mit Tests absichern	165
9 Hindernisse beim Anbringen von Tests	167
9.1 Abhängigkeiten	167
9.1.1 Statische Abhängigkeiten	167
9.1.2 Laufzeitabhängigkeiten	168
9.1.3 Überprüfung von Laufzeitabhängigkeiten vor der Programmausführung	168
9.1.4 Überprüfen von Laufzeitabhängigkeiten in der Produktion	173
9.1.5 Eingebettete Sprachen und öffentliche APIs	174
9.1.6 Abhängigkeiten, die die Erstellung von Tests behindern	174
9.2 Wo sollte man zuerst Tests anbringen?	175
9.3 Wenn Sie einen Fehler finden	176
10 Auflösen von Abhängigkeiten mithilfe von Testimplementierungen	177
10.1 Fallbeispiel: Sequenz-Generator	177
10.2 Verbessern der Testabdeckung durch Beobachter	183
11 Auflösen von Abhängigkeiten durch Ableitung (Object Seam)	189
11.1 Fallbeispiel: Authentifizierung an LDAP-Verzeichnis	189
11.2 Tests für die Fehlerbehandlung	199
12 Auflösen von Abhängigkeiten durch Beibehalten der Signatur (Link Seam)	203
12.1 Fallbeispiel: Java-Mail-API	203
12.2 Entfernen von unnötigen Duplikaten	215
13 Automatisches Generieren von Tests	221
13.1 Einweg-Generatoren	221
13.2 Regenerierende Generatoren	224
13.3 Round-Trip-Generatoren	225
13.4 Entwickeln von Testgeneratoren	225
Teil IV Refactoring bestehender Systeme	231
14 Erste Schritte hin zu besserem Code	233
14.1 Refactoring oder Reengineering?	233

14.2	Wo sollte man mit dem Refactoring beginnen?	234
14.3	Die »tägliche Hygiene«	234
14.4	Aufteilen großer Klassen	235
14.5	Zusammenfassen von Parametern durch Wertobjekte	238
15	Entfernen von Duplikaten	243
15.1	Entfernen von Duplikaten durch Vererbung	244
15.2	Entfernen von Duplikaten durch Delegation	247
15.3	Automatisches Aufspüren von Duplikaten	252
15.4	Duplikate außerhalb des Quellcodes	252
16	Aufteilung bedingter Logik	253
16.1	Umwandeln von Typcodes in Objekte	253
16.2	Aufteilen von Verteilern durch Kommandos	257
16.3	Aufteilen von Algorithmen durch Strategie	271
17	Refactoring von Logging und Fehlerbehandlung	285
17.1	Ersetzen von Logging durch Beobachter	285
17.2	Ersetzen mangelhafter Fehlerbehandlung durch Exceptions	292
18	Refactoring der Datenzugriffsschicht	301
18.1	Aufteilen von Active Records	301
18.2	Zusammenfassen von Finder-Methoden durch Filter	322
18.2.1	Einschub: Erweiterte Filter	339
Teil V Anhang		345
A.1	Software-Sanierung – Ein nicht zu unterschätzender Aufwand	345
A.1.1	Damit es nicht wieder soweit kommt	346
A.2	Referenzen	347
A.2.1	Einleitung	347
A.2.2	Teil 1 – Grundlagen	347
A.2.3	Teil 2 – Weiterentwicklung bestehender Systeme ohne vorhandene Tests	350
A.2.4	Teil 3 – Bestehende Systeme mit Tests absichern	350
A.2.5	Teil 4 – Refactoring bestehender Systeme	351
Stichwortverzeichnis		353