



Inhaltsverzeichnis

Sebastian Bergmann, Stefan Pribsch

Softwarequalität in PHP-Projekten

ISBN (Buch): 978-3-446-43539-1

ISBN (E-Book): 978-3-446-43582-7

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-43539-1>

sowie im Buchhandel.

Inhalt

Geleitwort	XV
Vorwort	XVII
Teil I Grundlagen	1
1 Softwarequalität	3
1.1 Was ist Softwarequalität?	3
1.2 Externe Qualität	4
1.3 Interne Qualität	4
1.4 Technische Schulden	5
1.5 Konstruktive Qualitätssicherung	7
1.6 Sauberer Code	8
1.6.1 Explizite und minimale Abhängigkeiten	9
1.6.2 Klare Verantwortlichkeiten	9
1.6.3 Keine Duplikation	9
1.6.4 Kurze Methoden mit wenigen Ausführungszweigen	9
1.7 Software-Metriken	9
1.7.1 Zyklomatische Komplexität und NPath-Komplexität	10
1.7.2 Change Risk Anti-Patterns (CRAP) Index	10
1.7.3 Non-Mockable Total Recursive Cyclomatic Complexity	11
1.7.4 Global Mutable State	11
1.7.5 Kohäsion und Kopplung	12
1.8 Werkzeuge	12
1.9 Fazit	14
2 Testen von Software	17
2.1 Einführung	17

2.2	Systemtests	19
2.2.1	Testen im Browser	19
2.2.2	Automatisierte Tests	20
2.2.3	Testisolation	22
2.2.4	Akzeptanztests	23
2.2.5	Grenzen von Systemtests	23
2.3	Unit-Tests	24
2.3.1	Rückgabewerte	26
2.3.2	Abhängigkeiten	28
2.3.3	Seiteneffekte	29
2.3.4	Stub- und Mock-Objekte	29
2.4	Die Softwaretestpyramide	31
2.5	Integrationstests	36
2.5.1	Kollaborierende Systeme ersetzen	37
2.5.2	Datenbanken ersetzen	38
2.5.3	Die GUI (zunächst) ignorieren	38
2.5.4	Frontend-Tests	39
2.5.5	PHPUnit als Infrastruktur	39
2.5.6	Realisierung	40
2.6	Fazit	52
3	Testen von Legacy Code	53
3.1	Einführung	53
3.2	Praxisbeispiel	55
3.2.1	Vorbereitungen	58
3.2.2	Globale Abhängigkeiten	62
3.2.3	Datenquellen	63
3.2.4	Asynchrone Vorgänge	69
3.2.5	Änderungen in der Datenbank	74
3.2.6	Nicht vorhersagbare Ergebnisse	75
3.2.7	Eingabedaten	78
3.2.8	Weiterführende Überlegungen	79
3.3	Fazit	80
Teil II	Fortgeschrittene Themen	81
4	Bad Practices in Unit-Tests	83
4.1	Einführung	83

4.2	Warum guter Testcode wichtig ist	83
4.3	Bad Practices und Test-Smells	84
4.3.1	Duplizierter Testcode	85
4.3.2	Zusicherungsroulette und begierige Tests	86
4.3.3	Fragile Tests	89
4.3.4	Obskure Tests	91
4.3.5	Lügende Tests	97
4.3.6	Langsame Tests	98
4.3.7	Konditionale Logik in Tests	100
4.3.8	Selbstvalidierende Tests	101
4.3.9	Websurfende Tests	102
4.3.10	Mock-Overkill	103
4.3.11	Skip-Epidemie	105
4.4	Fazit	105
5	Kontinuierliche Integration	107
5.1	Einführung	107
5.1.1	Kontinuierliche Integration	108
5.1.2	Statische Analyse	111
5.2	Installation und Inbetriebnahme	123
5.3	Konfiguration	123
5.3.1	Statische Tests	125
5.3.2	Dynamische Tests	131
5.3.3	Reporting	132
5.3.4	Deliverables erzeugen	133
5.4	Betrieb	135
5.5	Weiterführende Themen	135
5.5.1	Continuous Deployment	135
5.5.2	Einen Reverse Proxy nutzen	137
5.5.3	Kontinuierliche Integration und agile Paradigmen	137
5.6	Fazit	138
6	Testen von Datenbank-Interaktionen	141
6.1	Einführung	141
6.2	Pro und Kontra	142
6.2.1	Was gegen Datenbanktests spricht	142
6.2.2	Warum wir Datenbanktests schreiben sollten	143
6.3	Was wir testen sollten	144
6.4	Datenbanktests schreiben	145

6.4.1	Die Datenbankverbindung mocken	145
6.4.2	Die Datenbankerweiterung von PHPUnit	146
6.4.3	Die Klasse für Datenbanktestfälle.....	147
6.4.4	Die Verbindung zur Testdatenbank aufbauen	148
6.4.5	Datenbestände erzeugen	151
6.4.6	Operationen auf den Daten.....	166
6.4.7	Tests schreiben	169
6.4.8	Den Datenbanktester benutzen.....	176
6.5	Testgetriebene Entwicklung und Datenbanktests.....	179
6.6	Datenbanktests als Regressionstests	179
6.6.1	Probleme mit den Daten testen	180
6.6.2	Probleme testen, die durch Daten sichtbar werden	181
6.7	Fazit.....	182
7	Gebrauchstauglichkeit	183
7.1	Einführung.....	183
7.2	Anything goes – aber zu welchem Preis?.....	185
7.3	Designaspekte.....	186
7.3.1	Barrierefreiheit	186
7.3.2	Lesbarkeit	187
7.3.3	Label für Formularelemente	188
7.3.4	Tastaturbedienbare Webseite.....	188
7.3.5	Gute Farbkontraste	189
7.3.6	Logo zur Startseite verlinken	190
7.3.7	Alternativtexte für Bilder	190
7.3.8	Hintergrundbild mit Hintergrundfarbe	190
7.3.9	Druckversion nicht vergessen	190
7.3.10	Erkennbare Links.....	190
7.3.11	Gute Bookmarks.....	191
7.3.12	Keine Frames	191
7.3.13	Skalierbare Schrift.....	191
7.4	Technische Aspekte.....	192
7.4.1	Performanz.....	192
7.4.2	JavaScript.....	194
7.5	Benutzerführung.....	195
7.5.1	Der Mythos des Falzes	195
7.5.2	Feedback bei Interaktionen.....	196
7.5.3	Navigation.....	196

7.5.4	Popups und andere Störenfriede.....	197
7.5.5	Gewohnheiten bedienen, Erwartungen nicht enttäuschen.....	198
7.5.6	Fehlertoleranz und Feedback.....	199
7.6	Testen der Usability.....	199
7.7	Fazit.....	200
8	Performanz	203
8.1	Einführung.....	203
8.1.1	Werkzeuge.....	204
8.1.2	Umgebungsbezogene Gesichtspunkte	205
8.2	Lasttests	206
8.2.1	Apache Bench.....	207
8.2.2	Pylot.....	209
8.2.3	Weitere Werkzeuge für Lasttests	211
8.3	Profiling	212
8.3.1	Callgrind.....	213
8.3.2	APD	217
8.3.3	Xdebug.....	219
8.3.4	XHProf	219
8.3.5	OProfile	222
8.4	Systemmetriken.....	223
8.4.1	strace.....	223
8.4.2	Sysstat.....	224
8.4.3	Lösungen im Eigenbau	226
8.5	Übliche Fallstricke	227
8.5.1	Entwicklungsumgebung gegen Produktivumgebung.....	227
8.5.2	CPU-Zeit.....	227
8.5.3	Mikro-Optimierungen	228
8.5.4	PHP als <i>Glue Language</i>	228
8.5.5	Priorisierung von Optimierungen	229
8.6	Fazit.....	230
9	Sicherheit.....	231
9.1	Was ist eigentlich Sicherheit?	231
9.2	Secure by Design.....	232
9.2.1	Der Betrieb	232
9.2.2	Physikalischer Zugang.....	233
9.2.3	Software-Entwicklung	234
9.3	Was kostet Sicherheit?.....	237
9.4	Die häufigsten Probleme	238
9.5	Fazit.....	247

10 Testbasierte Entwicklung verkaufen	249
10.1 Vom prozeduralen Code zum testbasierten Vorgehen	249
10.2 Ziele der testbasierten Entwicklung	251
10.3 Aufwände für Software-Entwicklung	252
10.4 Möglichst wenige technische Schulden aufnehmen!	254
10.5 Offenlegung von Risiken mit ATAM	255
10.5.1 Diskutieren und entscheiden	258
10.5.2 Mit ATAM transparente Entscheidungen herbeiführen	258
10.6 Kalkulation testbasierter Entwicklung	258
10.6.1 Risiken als Argumentationshilfe berechnen	259
10.6.2 Langsamere Entwicklung bei höherer Qualität	259
10.6.3 Automatisierungs- und Abdeckungsgrad durch Tests bestimmen	261
10.7 Strategische Argumente für die Einführung testbasierter Entwicklung	262
10.7.1 Qualität und Nachhaltigkeit als Teil des Leistungsversprechens	262
10.7.2 Initiale Mehraufwände, die sich für den Auftraggeber lohnen	263
10.8 Das Angebot richtig verhandeln	264
10.9 Formulierung des Angebots	268
10.9.1 Inhalte des Angebots	269
10.9.2 Ein Angebot ohne Verhandlung abgeben?	269
10.10 Fazit	270
Teil III Fallstudien: Open-Source	271
11 TYPO3: die agile Zukunft eines schwergewichtigen Projekts	273
11.1 Einführung	273
11.1.1 Die Geschichte von TYPO3 – 13 Jahre in 13 Absätzen	273
11.1.2 Den Neuanfang wagen!	275
11.1.3 Unsere Erfahrungen mit dem Testen	276
11.2 Grundsätze und Techniken	277
11.2.1 Bittersüße Elefantentückchen	277
11.2.2 Testgetriebene Entwicklung	278
11.2.3 Tests als Dokumentation	279
11.2.4 Kontinuierliche Integration	280
11.2.5 Sauberer Code	281
11.2.6 Refaktorisierung	282
11.2.7 Programmierrichtlinien	283
11.2.8 Domänengetriebenes Design	285
11.3 Vorgehen bei der Entwicklung	285

11.3.1	Neuen Code entwickeln	286
11.3.2	Code erweitern und ändern	286
11.3.3	Code optimieren	287
11.3.4	Fehler finden und beheben	289
11.3.5	Alten Code fachgerecht entsorgen	289
11.4	Testrezepte	290
11.4.1	Ungewollt funktionale Unit-Tests	290
11.4.2	Zugriffe auf das Dateisystem	291
11.4.3	Konstruktoren in Interfaces	292
11.4.4	Abstrakte Klassen testen	293
11.4.5	Testen von geschützten Methoden	293
11.4.6	Verwendung von Callbacks	295
11.5	Auf in die Zukunft	297
12	Testen von Symfony und Symfony-Projekten	299
12.1	Einführung	299
12.2	Ein Framework testen	300
12.2.1	Der Release-Management-Prozess von Symfony	300
12.2.2	Verhältnis von Testcode und getestetem Code	302
12.2.3	Die Ausführung der Testsuite muss schnell sein	302
12.2.4	Gesammelte Erfahrungen	303
12.3	Testen von Webanwendungen	308
12.3.1	Die Hemmschwelle für das Testen abbauen	308
12.3.2	Unit-Tests	309
12.3.3	Funktionale Tests	314
12.4	Fazit	318
13	Testen von Grafikausgaben	319
13.1	Einführung	319
13.2	Entwicklungsphilosophie	320
13.3	Die ezcGraph-Komponente	320
13.3.1	Architektur	322
13.3.2	Anforderungen an die Tests	323
13.4	Ausgabetreiber durch Mock-Objekt ersetzen	323
13.4.1	Mehrfache Erwartungen	325
13.4.2	Structs	326
13.4.3	Generierung der Erwartungen	327
13.4.4	Zusammenfassung	328
13.5	Binäre Ausgaben testen	328

13.5.1	Die Ausgabetreiber.....	329
13.5.2	Generierung der Erwartungen	329
13.5.3	SVG	330
13.5.4	Bitmap-Erzeugung.....	331
13.5.5	Flash	334
13.6	Fazit.....	337
14	Testen von serviceorientierten APIs	339
14.1	Die Probleme	341
14.2	API-Zugangskennungen	342
14.3	API-Beschränkungen.....	345
14.4	Service-Protokolle offline testen.....	346
14.5	Konkrete Services offline testen	351
14.6	Fazit.....	356
15	Wie man einen WebDAV-Server testet	357
15.1	Über die eZ WebDAV-Komponente	357
15.1.1	WebDAV	357
15.1.2	Architektur	359
15.2	Herausforderungen bei der Entwicklung.....	361
15.2.1	Anforderungsanalyse	361
15.2.2	TDD nach RFC.....	362
15.2.3	Den Server testen	363
15.3	Automatisierte Akzeptanztests mit PHPUnit	365
15.3.1	Test-Trails aufzeichnen.....	367
15.3.2	Das Testrezept	368
15.3.3	Integration mit PHPUnit	369
15.4	Fazit.....	378
Teil IV	Fallstudien: Unternehmen	379
16	swoodo – eine wahrhaft agile Geschichte	381
16.1	Einführung.....	381
16.2	Evolution: Nur die Starken überleben	381
16.3	Wie wir die „eXtreme Seite“ erreichten	386
16.3.1	Kontinuierliche Integration.....	387
16.3.2	Testgetriebene Entwicklung	388
16.3.3	Tägliche Standup-Meetings	388
16.4	Wo wir schon einmal dabei sind	390

16.4.1	User Storys und Story Points	390
16.4.2	Velocity	391
16.4.3	Iterationsplanung	392
16.4.4	Programmieren in Paaren	392
16.4.5	Kollektives Eigentum	393
16.4.6	Offenheit für Änderungen	395
16.4.7	Überstunden	396
16.5	Die Kunst der Evolution	396
16.6	KISS und YAGNI – zwei Seiten einer Medaille	402
16.7	Evolutionstheorie und Fazit	402
17	Qualitätssicherung bei studiVZ	405
17.1	Einführung	405
17.2	Akzeptanztests	407
17.3	Selenium	408
17.3.1	Die Selenium-Erweiterung von PHPUnit	410
17.4	Technisches Setup von studiVZ	411
17.4.1	Codeumgebung	411
17.4.2	Testumgebung	412
17.5	Best Practices	413
17.5.1	Jugendsünden	413
17.5.2	Strategiewechsel	415
17.6	Eine DSL muss her	425
17.6.1	Interne DSL	426
17.6.2	Testing_SeleniumDSL 1.0	427
17.6.3	Testing_SeleniumDSL 2.0 – ein Entwurf	429
17.7	Fazit	431
18	Qualitätssicherung bei Digg	433
18.1	Die Ausgangssituation	433
18.1.1	Unsere Probleme	433
18.1.2	Code-Altlasten	434
18.1.3	Wie lösen wir unsere Probleme?	435
18.1.4	Ein Test-Framework wählen	437
18.1.5	Mit einem Experten arbeiten	438
18.2	Das Team trainieren	438
18.3	Testbaren Code schreiben	442
18.3.1	Statische Methoden vermeiden	442
18.3.2	Dependency Injection	445

18.4	Mock-Objekte	445
18.4.1	Überblick	445
18.4.2	Datenbank	445
18.4.3	Lose gekoppelte Abhängigkeiten	446
18.4.4	Beobachter für klasseninternes Verhalten	447
18.4.5	Memcache.....	449
18.4.6	Mocken einer serviceorientierten Architektur	450
18.5	Der Qualitätssicherungsprozess bei Digg	454
18.5.1	Testen	454
18.5.2	Vorteile.....	456
18.5.3	Herausforderungen	457
18.6	Fazit.....	458
	Schlussbetrachtungen	459
	Literatur	461
	Stichwortverzeichnis	467