

Inhaltsverzeichnis

I Grundlagen	1
1 Software-Qualität	3
1.1 Was ist Software-Qualität?	3
1.1.1 Externe Qualität	4
1.1.2 Interne Qualität	5
1.2 Technical Debt	6
1.3 Konstruktive Qualitätssicherung	8
1.4 Sauberer Code	9
1.4.1 Explizite und minimale Abhängigkeiten	9
1.4.2 Klare Verantwortlichkeiten	10
1.4.3 Keine Duplikation	10
1.4.4 Kurze Methoden mit wenigen Ausführungszweigen	10
1.5 Software-Metriken	10
1.5.1 Zyklomatische Komplexität und NPath-Komplexität	11
1.5.2 Change Risk Anti-Patterns (CRAP) Index	11
1.5.3 Non-Mockable Total Recursive Cyclomatic Complexity	12
1.5.4 Global Mutable State	12
1.5.5 Kohäsion und Kopplung	13
1.6 Werkzeuge	13
1.7 Fazit	16
2 Testen von Software	17
2.1 Black-Box- und White-Box-Tests	17
2.2 Wie viele Tests braucht man?	18
2.3 Systemtests	20
2.3.1 Testen im Browser	20

2.3.2	Automatisierte Tests	21
2.3.3	Testisolation	23
2.3.4	Akzeptanztests	24
2.3.5	Grenzen von Systemtests	25
2.4	Unit-Tests	25
2.4.1	Rückgabewerte	28
2.4.2	Abhängigkeiten	30
2.4.3	Seiteneffekte	31
2.5	Praxisbeispiel	32
2.5.1	Den zu testenden Code analysieren	36
2.5.2	Eine Testumgebung aufbauen	37
2.5.3	Globale Abhängigkeiten vermeiden	40
2.5.4	Unabhängig von Datenquellen testen	41
2.5.5	Asynchrone Vorgänge testen	49
2.5.6	Änderungen in der Datenbank speichern	54
2.5.7	Nicht vorhersagbare Ergebnisse	56
2.5.8	Eingabedaten kapseln	59
2.5.9	Weiterführende Überlegungen	61
2.6	Fazit	62
II	Best Practices	63
3	TYPO3: die agile Zukunft eines schwergewichtigen Projekts	65
3.1	Einführung	65
3.1.1	Die Geschichte von TYPO3 – 13 Jahre in 13 Absätzen	65
3.1.2	Den Neuanfang wagen!	67
3.1.3	Unsere Erfahrungen mit dem Testen	68
3.2	Grundsätze und Techniken	69
3.2.1	Bittersüße Elefantenstückchen	70
3.2.2	Testgetriebene Entwicklung	71
3.2.3	Tests als Dokumentation	72
3.2.4	Kontinuierliche Integration	73
3.2.5	Sauberer Code	74
3.2.6	Refaktorierung	76
3.2.7	Programmierrichtlinien	77

3.2.8 Domänengetriebenes Design	78
3.3 Vorgehen bei der Entwicklung	79
3.3.1 Neuen Code entwickeln	79
3.3.2 Code erweitern und ändern	80
3.3.3 Code optimieren	81
3.3.4 Fehler finden und beheben	83
3.3.5 Alten Code fachgerecht entsorgen	83
3.4 Testrezepte	84
3.4.1 Ungewollt funktionale Unit-Tests	84
3.4.2 Zugriffe auf das Dateisystem	85
3.4.3 Konstruktoren in Interfaces	86
3.4.4 Abstrakte Klassen testen	87
3.4.5 Testen von geschützten Methoden	88
3.4.6 Verwendung von Callbacks	91
3.5 Auf in die Zukunft	92
4 Bad Practices in Unit-Tests	95
4.1 Einführung	95
4.2 Warum guter Testcode wichtig ist	95
4.3 Bad Practices und Test-Smells	96
4.3.1 Duplizierter Testcode	97
4.3.2 Zusicherungsroulette und begierige Tests	99
4.3.3 Fragile Tests	102
4.3.4 Obskure Tests	104
4.3.5 Lügende Tests	112
4.3.6 Langsame Tests	113
4.3.7 Konditionale Logik in Tests	114
4.3.8 Selbstvalidierende Tests	116
4.3.9 Websurfende Tests	117
4.3.10 Mock-Overkill	119
4.3.11 Skip-Epidemie	120
4.4 Fazit	121
5 Qualitätssicherung bei Digg	123
5.1 Die Ausgangssituation	123
5.1.1 Unsere Probleme	123

5.1.2	Code-Altlasten	124
5.1.3	Wie lösen wir unsere Probleme?	126
5.1.4	Ein Test-Framework wählen	128
5.1.5	Mit einem Experten arbeiten	128
5.2	Das Team trainieren	129
5.3	Testbaren Code schreiben	133
5.3.1	Statische Methoden vermeiden	133
5.3.2	Dependency Injection	136
5.4	Mock-Objekte	137
5.4.1	Überblick	137
5.4.2	Datenbank	137
5.4.3	Lose gekoppelte Abhängigkeiten	138
5.4.4	Beobachter für klasseninternes Verhalten	139
5.4.5	Memcache	141
5.4.6	Mocken einer serviceorientierten Architektur	142
5.5	Der Qualitätssicherungsprozess bei Digg	147
5.5.1	Testen	147
5.5.2	Vorteile	149
5.5.3	Herausforderungen	151
5.6	Fazit	152
III	Server und Services	153
6	Testen von serviceorientierten APIs	155
6.1	Die Probleme	157
6.2	API-Zugangskennungen	158
6.3	API-Beschränkungen	163
6.4	Service-Protokolle offline testen	164
6.5	Konkrete Services offline testen	169
6.6	Fazit	175
7	Wie man einen WebDAV-Server testet	177
7.1	Über die eZ WebDAV-Komponente	177
7.1.1	WebDAV	177
7.1.2	Architektur	180

7.2 Herausforderungen bei der Entwicklung	182
7.2.1 Anforderungsanalyse	182
7.2.2 TDD nach RFC	183
7.2.3 Den Server testen	184
7.3 Automatisierte Akzeptanztests mit PHPUnit	186
7.3.1 Test-Trails aufzeichnen	188
7.3.2 Das Testrezept	190
7.3.3 Integration mit PHPUnit	192
7.4 Fazit	201
IV Architektur	203
8 Testen von Symfony und Symfony-Projekten	205
8.1 Einführung	205
8.2 Ein Framework testen	206
8.2.1 Der Release-Management-Prozess von Symfony	206
8.2.2 Verhältnis von Testcode und getestetem Code	208
8.2.3 Die Ausführung der Testsuite muss schnell sein	208
8.2.4 Gesammelte Erfahrungen	209
8.3 Testen von Webanwendungen	215
8.3.1 Die Hemmschwelle für das Testen abbauen	215
8.3.2 Unit-Tests	216
8.3.3 Funktionale Tests	222
8.4 Fazit	227
9 Testen von Grafikausgaben	229
9.1 Einführung	229
9.2 Entwicklungsphilosophie	230
9.3 Die ezcGraph-Komponente	230
9.3.1 Architektur	232
9.3.2 Anforderungen an die Tests	233
9.4 Ausgabetreiber durch Mock-Objekt ersetzen	234
9.4.1 Mehrfache Erwartungen	236
9.4.2 Structs	238
9.4.3 Generierung der Erwartungen	238

9.4.4	Zusammenfassung	239
9.5	Binäre Ausgaben testen	239
9.5.1	Die Ausgabetreiber	240
9.5.2	Generierung der Erwartungen	241
9.5.3	SVG	241
9.5.4	Bitmap-Erzeugung	243
9.5.5	Flash	246
9.6	Fazit	249
10	Testen von Datenbank-Interaktionen	251
10.1	Einführung	251
10.2	Pro und Kontra	252
10.2.1	Was gegen Datenbanktests spricht	252
10.2.2	Warum wir Datenbanktests schreiben sollten	253
10.3	Was wir testen sollten	254
10.4	Datenbanktests schreiben	256
10.4.1	Die Datenbankverbindung mocken	256
10.4.2	Die Datenbankerweiterung von PHPUnit	257
10.4.3	Die Klasse für Datenbanktestfälle	258
10.4.4	Die Verbindung zur Testdatenbank aufbauen	259
10.4.5	Datenbestände erzeugen	263
10.4.6	Operationen auf den Daten	280
10.4.7	Tests schreiben	283
10.4.8	Den Datenbanktester benutzen	292
10.5	Testgetriebene Entwicklung und Datenbanktests	294
10.6	Datenbanktests als Regressionstests	295
10.6.1	Probleme mit den Daten testen	296
10.6.2	Probleme testen, die durch Daten sichtbar werden	297
10.7	Zusammenfassung	298
V	QA im Großen	299
11	Qualitätssicherung bei studiVZ	301
11.1	Einführung	301
11.2	Akzeptanztests	303

11.3 Selenium	305
11.3.1 Die Selenium-Erweiterung von PHPUnit	307
11.4 Technisches Setup von studiVZ	308
11.4.1 Codeumgebung	308
11.4.2 Testumgebung	309
11.5 Best Practices	310
11.5.1 Jugendsünden	310
11.5.2 Strategiewechsel	313
11.6 Eine DSL muss her	326
11.6.1 Interne DSL	327
11.6.2 Testing_SeleniumDSL 1.0	327
11.6.3 Testing_SeleniumDSL 2.0 – ein Entwurf	330
11.7 Fazit	331
12 Kontinuierliche Integration	335
12.1 Einführung	335
12.1.1 Kontinuierliche Integration	337
12.1.2 Statische Analyse	339
12.2 Installation und Inbetriebnahme	353
12.3 Konfiguration	353
12.3.1 Statische Tests	356
12.3.2 Dynamische Tests	363
12.3.3 Reporting	363
12.3.4 Deliverables erzeugen	365
12.4 Betrieb	366
12.5 Weiterführende Themen	367
12.5.1 Continuous Deployment	367
12.5.2 Einen Reverse Proxy nutzen	369
12.5.3 Kontinuierliche Integration und agile Paradigmen	369
12.6 Fazit	370
13 swoodoo – eine wahrhaft agile Geschichte	373
13.1 Einführung	373
13.2 Evolution: Nur die Starken überleben	374
13.3 Wie wir die "eXtreme Seite" erreichten	379
13.3.1 Kontinuierliche Integration	380

13.3.2 Testgetriebene Entwicklung	381
13.3.3 Tägliche Standup-Meetings	382
13.4 Und wo wir schon einmal dabei sind	383
13.4.1 User Storys und Story Points	384
13.4.2 Velocity	385
13.4.3 Iterationsplanung	386
13.4.4 Programmieren in Paaren	386
13.4.5 Kollektives Eigentum	388
13.4.6 Offenheit für Änderungen	389
13.4.7 Überstunden	391
13.5 Die Kunst der Evolution	391
13.6 KISS und YAGNI – zwei Seiten einer Medaille	397
13.7 Evolutionstheorie und Fazit	398
VI Nichtfunktionale Aspekte	401
14 Gebrauchstauglichkeit	403
14.1 Einführung	403
14.2 Anything goes – aber zu welchem Preis?	405
14.3 Designaspekte	407
14.3.1 Barrierefreiheit	407
14.3.2 Lesbarkeit	407
14.3.3 Label für Formularelemente	408
14.3.4 Tastaturbedienbare Webseite	408
14.3.5 Gute Farbkontraste	410
14.3.6 Logo zur Startseite verlinken	410
14.3.7 Alternativtexte für Bilder	410
14.3.8 Hintergrundbild mit Hintergrundfarbe	410
14.3.9 Druckversion nicht vergessen	411
14.3.10 Erkennbare Links	411
14.3.11 Gute Bookmarks	411
14.3.12 Keine Frames	412
14.3.13 Skalierbare Schrift	412
14.4 Technische Aspekte	412
14.4.1 Performanz	412

14.4.2 JavaScript	414
14.5 Benutzerführung	415
14.5.1 Der Mythos des Falzes	416
14.5.2 Feedback bei Interaktionen	417
14.5.3 Navigation	417
14.5.4 Popups und andere Störenfriede	418
14.5.5 Gewohnheiten bedienen, Erwartungen nicht enttäuschen	419
14.5.6 Fehlertoleranz und Feedback	419
14.6 Testen der Usability	420
14.7 Fazit	421
15 Performanz	423
15.1 Einführung	423
15.1.1 Werkzeuge	424
15.1.2 Umgebungsbezogene Gesichtspunkte	425
15.2 Lasttests	427
15.2.1 Apache Bench	428
15.2.2 Pylot	430
15.2.3 Weitere Werkzeuge für Lasttests	432
15.3 Profiling	432
15.3.1 Callgrind	434
15.3.2 APD	439
15.3.3 Xdebug	440
15.3.4 XHProf	442
15.3.5 OProfile	443
15.4 Systemmetriken	445
15.4.1 strace	445
15.4.2 Sysstat	446
15.4.3 Lösungen im Eigenbau	448
15.5 Übliche Fallstricke	449
15.5.1 Entwicklungsumgebung gegen Produktivumgebung	449
15.5.2 CPU-Zeit	450
15.5.3 Mikro-Optimierungen	450
15.5.4 PHP als <i>Glue Language</i>	451
15.5.5 Priorisierung von Optimierungen	451
15.6 Fazit	452

16 Sicherheit	455
16.1 Was ist eigentlich Sicherheit?	455
16.2 Secure by Design	456
16.2.1 Der Betrieb	456
16.2.2 Physikalischer Zugang	458
16.2.3 Software-Entwicklung	458
16.3 Was kostet Sicherheit?	462
16.4 Die häufigsten Probleme	463
16.4.1 A10 – Transportschicht	463
16.4.2 A9 – Kryptografie	464
16.4.3 A8 – Weiterleitungen	465
16.4.4 A7 – Zugriffsschutz	466
16.4.5 A6 – Konfiguration	467
16.4.6 A5 – Cross Site Request Forgery (CSRF/XSRF)	468
16.4.7 A4 – Direkte Zugriffe	468
16.4.8 A3 – Authentifizierung und Session-Management	469
16.4.9 A2 – Cross Site Scripting (XSS)	470
16.4.10 A1 – Injections	472
16.5 Fazit	473
VII Schlussbetrachtungen	475
Stichwortverzeichnis	489