

Inhaltsverzeichnis

I	Warum überhaupt testen?	1
1	Komplexe Systeme führen zu Fehlern	3
1.1	Kommunikation	3
1.2	Gedächtnis	6
1.3	Fachlichkeit	6
1.4	Komplexität	7
1.5	Erstes Fazit	8
2	Programmiersprachen sind fehleranfällig	9
2.1	Die Venussonde Mariner 1	9
2.2	Der Jungfernflug der Ariane 5	10
2.3	Zweites Fazit	12
3	Qualität, Fehler, Test: Versuch einer Begriffsbestimmung	15
3.1	Qualität	15
3.2	Anforderung	17
3.3	Fehler	18
3.4	Test	20
3.4.1	Demonstratives und destruktives Testen	20
3.4.2	White-Box- und Black-Box-Testverfahren	21
3.4.3	Testfall und Testdaten	22
3.4.4	Unit-Tests: Klassen-, Ketten- und Modultests	22
3.4.5	Debugging	25
4	Schlussbemerkungen	27
II	Verfahren des Softwaretests	29
5	Lösungen für technische Probleme	31
5.1	Unterstützung durch den Compiler	31
5.1.1	Warninglevel	31
5.1.2	Programmierrichtlinien	31
5.2	Was nützen statische strenge Typprüfungen?	41

5.3	Debugging	42
5.3.1	Einplanung der Fehlersuche in Produkt und Prozess	42
5.3.2	Vorbereitung und Ausführung des Debugging	42
5.3.3	Der Debugging-Vorgang	44
6	Lösungen für analytische Probleme	47
6.1	Scope: Was will ich testen?	47
6.2	Fachliche Testfälle finden	48
6.2.1	Testdaten ableiten	51
6.2.2	Unit-Testfälle ableiten	52
6.2.3	Kettentests ableiten	52
6.2.4	System-Testfälle ableiten	53
7	Lösungen für methodische Probleme	55
7.1	Psychologie des Testens	55
7.2	Codereviews	57
7.2.1	Interne Codereviews	58
7.2.2	Externe Codereviews	59
7.2.3	Dokumentreviews	59
7.3	Die richtigen Testdaten finden	60
7.3.1	Grenz- und Extremwerte	60
7.3.2	Testdaten als Designkriterium	63
7.3.3	Fehlersensibilität	64
7.3.4	Äquivalenzklassen	65
7.4	Überdeckungen: Wege durch die kombinatorische Explosion	69
7.4.1	Anweisungs-, Zweig- und Pfadüberdeckung	69
7.4.2	Vereinfachte Schleifenüberdeckung	72
7.4.3	Test von Bedingungen: die Termüberdeckung	73
7.5	Unbezahlbare Erfahrung: Error Guessing und laterale Tests	74
8	Lösungen für fortgeschrittene Probleme	77
8.1	Zustandsraumbasiertes Testen	77
8.2	Rekursion und Nebenläufigkeit	81
8.2.1	Rekursive und iterative Algorithmen	81
8.2.2	Parallele Prozesse	85
9	Lösungen zum Test objektorientierter Software	93
9.1	Testreihenfolge in objektorientierten Programmen	95
9.1.1	Assoziationen	95
9.1.2	Vererbung	96
9.1.3	Testreihenfolge bei Verflechtung von Assoziationen und Vererbung ..	99
9.1.4	Testreihenfolgen für Methoden	100
9.2	Vererbung, das zweischneidige Schwert	101
9.2.1	Prinzipien zur objektorientierten Vererbung	104

9.2.2	Flattening: Welche Methoden sind zu testen?	106
9.2.3	Zufällige Korrektheit durch Vererbung	107
9.2.4	Typische Fehler in Vererbungshierarchien	108
9.2.5	Teststrategie bei Vererbung	110
9.3	Testmuster: Tipps für die Praxis	111
9.3.1	Modale Klasse	112
9.3.2	Modale Hierarchie	114
9.3.3	Nicht-modaler, polymorpher Server	117
9.4	Struktur von objektorientierten Programmen	119
9.5	Zusammenfassung	123

10 Lösungen für organisatorische Probleme 125

10.1	Testgetriebenes Design: Abläufe und Ausnahmen	125
10.1.1	Vorgehensweisen: Wasserfall oder Iterationen?	126
10.1.2	Testgetriebenes Design	139
10.2	Refactoring	143
10.2.1	Was ist Refactoring?	143
10.2.2	Wie funktioniert Refactoring?	144
10.3	Testkoordination	147
10.4	Aufwandsbetrachtungen	147
10.4.1	Schätzungen	150
10.4.2	Fehlerkorrekturen und Re-Tests	150
10.4.3	Fehlermodelle als Rechenmodelle zur Aufwandsschätzung	151
10.5	Testverwaltung	155

III Umsetzung in die Praxis 157

11 Automatisierung von Entwicklertests 159

11.1	Testfall-Findung vs. Testfall-Automatisierung	159
11.1.1	Testautomatisierung und testgetriebenes Vorgehen	159
11.1.2	Anforderungen an die Testautomatisierung	160
11.2	Das Konzept der xUnit-Familie	160
11.3	Entwicklertests mit xUnit	162
11.3.1	Design for Testability	163
11.3.2	JUnit	165
11.3.3	CppUnit	173
11.3.4	NUnit – JUnit unter .NET	176
11.3.5	Stellvertreterobjekte – Stub, Dummy und Mock	180
11.4	Drei JUnit-Testbeispiele	181
11.4.1	Komplettes Syntaxbeispiel	181
11.4.2	Grenz- und Extremwerte für einen Prüfmethode-Test	183
11.4.3	Test eines Zustandsautomaten mit einem Mock-Objekt	184

11.5	Testautomatisierung über die GUI	191
11.5.1	Lineare Skripte	192
11.5.2	Strukturierte Skripte	193
11.5.3	Verteilte Skripte	193
11.5.4	Datengetriebene Skripte	194
11.5.5	Schlüsselwortgetriebene Skripte	194
11.5.6	GUI-Tests mit JUnit	194
11.6	Stresstest-Automatisierung	195
11.7	Test von Mehrschicht-Anwendungen	196
11.8	Fehlerinjektion: Wie gut sind unsere Tests?	197
11.9	Mehrwert automatisierter Tests	198
12	Was haben wir aus der Betrachtung der Verfahren gelernt?	199
12.1	Kriterien für erfolgreiche Projekte	199
12.2	Anforderungen an das Entwicklungsteam	201
12.3	Anforderungen an den Projektleiter	202
13	Teststrategie: Der Weg ist wichtiger als das Ziel	205
13.1	Strategien umsetzen	205
13.2	Inhalte einer pragmatischen Entwicklertest-Strategie	208
14	Fehlerkultur	213
14.1	Konstruktive Fehlerkultur: aus Fehlern lernen	214
14.2	Fehlerkultur und Kreativität	215
14.3	Beurteilung und Konsequenzen von Fehlern	216
IV	Möglichkeiten und Herausforderungen	219
15	Test von Realtime und Embedded Systems	221
15.1	Was bedeutet eigentlich RTES?	221
15.2	Was ist ein sicheres System?	224
15.3	Warum sind RTES so besonders schwierig?	225
15.3.1	Reaktives System	225
15.3.2	Nebenläufigkeit und Verteilung	225
15.4	Besondere Testverfahren	226
15.4.1	Failure Mode and Effect Analysis – FMEA	227
15.4.2	Fault Tree Analysis – FTA	228
15.4.3	Classification Tree Method – CTM	228
15.4.4	Testbare und robuste Architektur	231
15.4.5	Gemischte Signale und Timing-Diagramme	232

16	UML 1.5 vs. UML 2.0	239
16.1	Aktivitätsdiagramme in der UML 2	239
16.2	Das Testprofil	245
16.2.1	Was ist ein UML-Profil?	246
16.2.2	Wie sieht das UML-Testprofil aus?	247
16.2.3	Ein Anwendungsbeispiel	250
16.3	Abbildung des UML-Testprofils auf JUnit	254
17	Zusammenwachsen von Entwicklung und Qualitätssicherung ...	257
17.1	Ziele für Entwicklung und Qualitätssicherung	257
17.2	Aufgabenteilung zwischen Entwicklung und Qualitätssicherung	258
 Anhang		 261
A	Beispiele für JUnit-Tests	261
A.1	Ein einfaches Testbeispiel	261
A.2	Grenz- und Extremwerte testen	263
A.3	Modale Klasse mit Mock testen	267
B	Beispiel für eine JUnit-Testsuite	283
C	Beispiel eines CppUnit-Tests	285
D	Beispiel eines NUnit-Tests in C#	295
E	Beispiel eines Jellytool-Tests	297
F	Übersicht aller 37 objektorientierten Testmuster	299
Glossar	303
Abbildungsverzeichnis	309
Tabellenverzeichnis	313
Codebeispielverzeichnis	315
Literaturverzeichnis	317
Kolophon	323
Danksagung	325
Index	327