

Der Inhalt (im Überblick)

	Einführung	xxiii
1	Gute Software entwickeln: <i>Den Kunden zufrieden stellen</i>	1
2	Anforderungen sammeln: <i>Wissen, was der Kunde will</i>	29
3	Projektplanung: <i>Mit Planung zum Erfolg</i>	69
4	User-Stories und Tasks: <i>Jetzt aber an die Arbeit</i>	109
5	Gut genug-Design: <i>Fertig werden – mit gutem Design</i>	149
6	Versionskontrolle: <i>Defensives Programmieren</i>	177
6.5	Code erstellen: <i>Lasche A durch Schlitz B stecken ...</i>	219
7	Tests und kontinuierliche Integration: <i>Damit nichts schiefgeht</i>	235
8	Testgetriebene Entwicklung: <i>Ihren Code verantwortlich machen</i>	275
9	Eine Iteration abschließen: <i>Alles fügt sich zusammen</i>	317
10	Die nächste Iteration: <i>Es ist nicht kaputt ... reparieren Sie es trotzdem</i>	349
11	Bugs: <i>Bugs beheben wie ein Profi</i>	383
12	Das wahre Leben: <i>Ein Prozess fürs Leben</i>	417

Der Inhalt (jetzt ausführlich)

Einführung

Ihr entwicklungsfähiges Gehirn. Sie versuchen, etwas zu lernen, und Ihr Hirn tut sein Bestes, damit das Gelernte nicht hängen bleibt. Es denkt nämlich: »Wir sollten lieber ordentlich Platz für wichtigere Dinge lassen, z.B. für das Wissen, welche Tiere einem gefährlich werden könnten, oder dass es eine ganz schlechte Idee ist, nackt Snowboard zu fahren.« Tja, wie schaffen wir es nun, Ihr Gehirn davon zu überzeugen, dass Ihr Leben davon abhängt, gute Software entwickeln zu können?

Für wen ist dieses Buch?	xxiv
Wir wissen, was Sie gerade denken.	xxv
Metakognition: Nachdenken übers Denken	xxvii
Und das können SIE tun, um sich Ihr Gehirn untertan zu machen	xxix
Lies mich!	xxx
Die Gutachter	xxxii
Danksagungen	xxxiii

Toms Touren gehen online	2
Die meisten Projekte starten mit zwei wichtigen Fragen	3
Entwicklung nach dem »Big Bang«-Ansatz	4
Wir spulen vor: zwei Wochen später	5
Big Bang-Entwicklung endet meist mit einer üblen BAUCHLANDUNG	6
Gute Software entwickeln heißt ...	9
Das Ziel mit ITERATIONEN erreichen	10
Jede Iteration ist ein Mini-Projekt	14
Jede Iteration entspricht QUALITÄTS-Software	14
Der Kunde WIRD etwas verändern	20
Es ist Ihre Aufgabe, Anpassungen vorzunehmen	20
Da gibt es aber ein paar MÄCHTIGE Probleme ...	20
Iteration berücksichtigt Veränderungen automatisch (irgendwie jedenfalls)	22
Ihre Software ist erst fertig, wenn sie VERÖFFENTLICHT ist	25
Werkzeuge für Ihre Softwareentwickler-Kiste	26

Ikarus Shuttle-Trips wird modernisiert	30
Sprechen Sie mit Ihrem Kunden, um an MEHR Informationen zu kommen	33
Blueskying – mit dem Kunden Wolken schieben	34
Manchmal endet Blueskying so ...	36
Finden Sie heraus, wie sich die Leute KONKRET verhalten	37
Ihre Anforderungen müssen KUNDENZentriert sein	39
Kundenfeedback zum Entwickeln von Anforderungen nutzen	41
User-Stories definieren das WAS Ihres Projekts ...	43
Abschätzungen definieren das WANN	43
Bürogespräch	47
Spielen wir das Planungsspiel	48
Machen Sie Annahmen den Prozess	51
Für eine User-Story ist eine GROSSE Abschätzung eine SCHLECHTE Abschätzung	54
Das Ziel heißt Konvergenz	57
Der Iterationszyklus von der Anforderung bis zur Abschätzung	60
Jetzt endlich sind Sie so weit und können das gesamte Projekt abschätzen ...	63

Toms Touren gehen online	2
Die meisten Projekte starten mit zwei wichtigen Fragen	3
Entwicklung nach dem »Big Bang«-Ansatz	4
Wir spulen vor: zwei Wochen später	5
Big Bang-Entwicklung endet meist mit einer üblen BAUCHLANDUNG	6
Gute Software entwickeln heißt ...	9
Das Ziel mit ITERATIONEN erreichen	10
Jede Iteration ist ein Mini-Projekt	14
Jede Iteration entspricht QUALITÄTS-Software	14
Der Kunde WIRD etwas verändern	20
Es ist Ihre Aufgabe, Anpassungen vorzunehmen	20
Da gibt es aber ein paar MÄCHTIGE Probleme ...	20
Iteration berücksichtigt Veränderungen automatisch (irgendwie jedenfalls)	22
Ihre Software ist erst fertig, wenn sie VERÖFFENTLICHT ist	25
Werkzeuge für Ihre Softwareentwickler-Kiste	26

Ikarus Shuttle-Trips wird modernisiert	30
Sprechen Sie mit Ihrem Kunden, um an MEHR Informationen zu kommen	33
Blueskying – mit dem Kunden Wolken schieben	34
Manchmal endet Blueskying so ...	36
Finden Sie heraus, wie sich die Leute KONKRET verhalten	37
Ihre Anforderungen müssen KUNDENZentriert sein	39
Kundenfeedback zum Entwickeln von Anforderungen nutzen	41
User-Stories definieren das WAS Ihres Projekts ...	43
Abschätzungen definieren das WANN	43
Bürogespräch	47
Spielen wir das Planungsspiel	48
Machen Sie Annahmen den Prozess	51
Für eine User-Story ist eine GROSSE Abschätzung eine SCHLECHTE Abschätzung	54
Das Ziel heißt Konvergenz	57
Der Iterationszyklus von der Anforderung bis zur Abschätzung	60
Jetzt endlich sind Sie so weit und können das gesamte Projekt abschätzen ...	63

Kunden wollen ihre Software SOFORT!	70
Prioritäten festlegen – mit dem Kunden zusammen	73
Wir wissen, was in Meilenstein 1.0 hineinkommt (ja, vielleicht)	74
Wenn die Features nicht passen, überarbeiten Sie die Prioritäten	75
Manchmal bedeuten mehr Leute weniger Ertrag	77
Arbeiten Sie sich zu einem vernünftigen Meilenstein 1.0 vor	78
Bei Iterationen liegt die Würze in der Kürze	85
Vergleich von Planung und Realität	87
Velocity berücksichtigt Overhead in Ihren Abschätzungen	89
Programmierer rechnen in UTOPISCHEN Tagen ...	90
Entwickler rechnen in REALEN Tagen ...	91
Wann dauert Ihre Iteration zu lange?	92
Befassen Sie sich mit dem Durchsatz, BEVOR Sie Iterationen einteilen	93
Zeit für eine Beurteilung	97
Mit völlig ungenutzten Kunden umgehen	98
Die große Tafel da an Ihrer Wand	100
Wie Sie den Leuten in Ihrem Team das Leben vermiesen	103

Dürfen wir vorstellen? – binHinUndWeg	110
Addieren sich Ihre Tasks zum richtigen Ergebnis?	113
Tragen Sie nur die Arbeit ab, die noch übrig ist	115
Hängen Sie Ihre Tasks an die Tafel	116
Fangen Sie an, an Ihren Tasks zu arbeiten	118
Ein Task ist erst dann IN ARBEIT, wenn er in Arbeit ist	119
Und wenn ich an zwei Sachen gleichzeitig arbeite?	120
Ihr erstes Stand-up-Meeting ...	123
Task 1: die »Date«-Klasse anlegen	124
Stand-up-Meeting: Tag 5, also Ende von Woche 1 ...	130
Stand-up-Meeting: Tag 2, Woche 2	136
Wir unterbrechen dieses Kapitel ...	140
Ungeplante Tasks müssen festgehalten werden	141
Unerwartete Tasks lassen Ihren Burndown-Verlauf ansteigen	143
Durchsatz ist hilfreich, aber ...	144
Sie haben eine Menge zu tun ...	146
... aber Sie wissen GENAU, wo Sie stehen	147
Durchsatz im Gespräch	148

binHinUndWeg steckt in ernsten Schwierigkeiten ...	150
Dieses Design verstößt gegen das Prinzip der einen Verantwortlichkeit	153
Mehrere Verantwortlichkeiten im Design aufspüren	156
Von mehreren Verantwortlichkeiten zu einer Verantwortlichkeit übergehen	159
Ihr Design sollte dem SRP folgen, aber ebenso dem DRY-Prinzip ...	160
Das Post-Refactoring-Stand-up-Meeting ...	164
Ungeplante Tasks sind trotz allem immer noch Tasks	166
Die Demo ist selbst Teil Ihres Tasks	167
Wenn alles erledigt ist, ist die Iteration beendet	170

Sie haben einen neuen Auftrag – BeatBox Pro	178
Und jetzt die Arbeit am GUI ...	182
Und ein schneller Test ...	183
Führen Sie Ihrem Kunden die neue BeatBox vor	185
Kommen wir zum ThemaVERSIONSKONTROLLE	188
Zuerst legen Sie Ihr Projekt an ...	190
... anschließend können Sie Code ein- und auschecken	191
Die meisten Werkzeuge zur Versionskontrolle versuchen, Probleme für Sie zu lösen	192
Der Server versucht, Ihre Änderungen ZUSAMMENZUFÜHREN	193
Wenn Ihre Software die Änderungen nicht zusammenführen kann, meldet sie einen Konflikt	194
Noch mehr Iterationen, noch mehr Stories ...	198
Wir haben mehr als eine Version von unserer Software ...	200
Gute Commit-Kommentare machen es Ihnen leichter, ältere Software zu finden	202
Jetzt können Sie Version 1.0 auschecken	203
(Notfall-)Stand-up-Meeting	204
Geben Sie Ihren Versionen ein Tag	205
Tags, Zweige und Trunks ... au weia!	207
Version 1.0 korrigieren ... diesmal aber wirklich!	208
Wir haben jetzt ZWEI Codestämme.	209
Wann man KEINEN Zweig anlegen sollte ...	212
Zen oder die Kunst vom guten Verzweigen	212
Was Versionskontrolle macht ...	214
Versionskontrolle kann nicht dafür sorgen, dass Ihr Code wirklich funktioniert ...	215
Werkzeuge für Ihre Softwareentwickler-Kiste	216

Entwickler sind keine Gedankenleser	220
Ihr Projekt in einem Schritt erstellen	221
Ant: Ein Build-Tool für Java-Projekte	222
Projekte, Properties, Targets, Tasks	223
Gute Build-Skripten ...	228
Gute Build-Skripten gehen über das Übliche HINAUS	230
Auch Ihr Build-Skript ist Code	232
Neuer Entwickler, Klappe, die zweite	233
Werkzeuge für Ihre Softwareentwickler-Kiste	234

Schiefgehen tut es IMMER	236
Ihr System können Sie auf dreierlei Weise betrachten	238
Black-Box-Tests konzentrieren sich auf EINGABE und AUSGABE	239
Grey-Box-Tests bringen Sie NÄHER an den Code	240
White-Box-Tests nutzen Insiderwissen	243
ALLES in einem Schritt testen	248
Tests mit einem Test-Framework automatisieren	250
Führen Sie Ihre Tests mit dem Framework aus	251
CI im Griff mit CruiseControl	254
Tests stellen sicher, dass alles funktioniert ... oder?	256
Den gesamten Code testen heißt, ALLE ZWEIGE zu testen	264
Mit einem Abdeckungsbericht prüfen, was abgedeckt wird	265
Eine gute Abdeckung zu erreichen ist nicht immer leicht ...	267
Was Versionskontrolle macht ...	270
Werkzeuge für Ihre Softwareentwickler-Kiste	274

Testen Sie ERST, nicht zum Schluss	276
Wir werden also ERST testen ...	277
Willkommen bei der testgesteuerten Entwicklung	277
Ihr erster Test ...	278
... schlägt kläglich fehl	279
Machen Sie Ihre Tests GRÜN	280
Rot, grün, Refactoring ...	281
Bei TDD LENKEN Tests Ihre Implementierung	286
Einen Task abzuschließen bedeutet, dass Sie alle erforderlichen Tests haben und diese alle erfolgreich sind	288
Gehen Sie weiter, wenn Ihr Test erfolgreich ist!	289
Einfachheit heißt, Abhängigkeiten zu vermeiden	293
Schreiben Sie immer testbaren Code	294
Schauen Sie sich Ihren Entwurf an, wenn das Testen schwer wird	295
Das Strategy-Muster ermöglicht mehrere Implementierungen eines Interface	296
Halten Sie Ihren Testcode bei Ihren Tests	299
Testen führt zu besserem Code	300
Mehr Tests bedeuten immer viel mehr Code	302
Strategy-Muster, lockere Bindung, Mock-Objekte ...	303
Wir brauchen viele unterschiedliche, aber ähnliche Objekte	304
Können wir nicht Objekte generieren?	304
Ein Mock-Objekt vertritt echte Objekt	305
Mock-Objekte sind funktionierende Objektvertreter	306
Gute Software ist testbar	309
Es ist nicht immer leicht, grün zu sein ...	310
Ein Tag im Leben eines testgesteuerten Entwicklers	312
Werkzeuge für Ihre Softwareentwickler-Kiste	314

Ihre Iteration ist fast abgeschlossen ...	318
... aber es gibt eine Menge, was Sie noch tun könnten	319
Systemtests sind NOTWENDIG ...	324
... aber WER führt sie durch?	325
Systemtests benötigen ein vollständiges System	326
Gute Systemtests erfordern ZWEI Iterationszyklen	327
Mehr Iterationen führen zu mehr Problemen	328
10 Kennzeichen effektiver Systemtests	333
Leben (und Sterben) eines Bugs	334
Sie haben also einen Bug gefunden ...	336
Anatomie eines Bug-Reports	337
Aber es gibt eine Menge Dinge, die Sie noch tun KÖNNTEN ...	338
Zeit für den Iterationsrückblick	342
Einige Fragen für den Iterationsrückblick	343
Eine ALLGEMEINE Prioritätsliste zur Erledigung ZUSÄTZLICHER Dinge ...	344
Werkzeuge für Ihre Softwareentwickler-Kiste	346

Was ist funktionierende Software?	350
Sie müssen die nächste Iteration planen	352
Der Durchsatz trägt dem WAHREN LEBEN Rechnung	359
Es geht IMMER noch um den Kunden	360
Auch die Software eines anderen ist BLOSS Software	362
Kundenzustimmung? Prüfen!	365
Ihren Code testen	370
Houston, wir haben ein ernsthaftes Problem ...	371
Vertrauen Sie KEINEM	373
Sie ohne Ihren Prozess	378
Sie mit Ihrem Prozess	379

Neulich in Iteration 2	386
Zunächst müssen Sie mit dem Kunden reden	386
Priorität eins: Die Dinge erstellbar machen	392
Wir könnten den Code reparieren ...	394
... müssen aber die Funktionalität reparieren	395
Herausfinden, welche Funktionalität funktioniert	396
JETZT wissen Sie, was nicht funktioniert	399
Was würden Sie tun?	399
Spike-Test zum Abschätzen	400
Was sagen Ihnen die Ergebnisse der Spike-Tests?	402
Das Bauchgefühl Ihres Teams zählt	404
Geben Sie Ihrem Kunden die Bug-Fix-Abschätzung	406
Die Dinge sehen gut aus ...	410
... und Sie schließen die Iteration erfolgreich ab!	411
UND der Kunde ist zufrieden	412
Werkzeuge für Ihre Softwareentwickler-Kiste	414

Einen Softwareentwicklungsprozess fixieren	418
Ein guter Prozess liefert gute Software aus	419
Förmliche Kleidung erforderlich ...	424
Einige zusätzliche Ressourcen ...	426
Mehr Wissen = bessere Prozesse	427
Werkzeuge für Ihre Softwareentwickler-Kiste	428

1. UML-Klassendiagramme	434
2. Sequenzdiagramme	436
3. User-Stories und Anwendungsfälle	438
4. System-Tests vs. Unit-Tests	440
5. Refactoring	441

Entwicklungstechniken	444
Entwicklungsprinzipien	446