Frank Witte

Testmanagement und Softwaretest

Theoretische Grundlagen und praktische Umsetzung

Springer Vieweg

Testmanagement und Softwaretest

Frank Witte

Testmanagement und Softwaretest

Theoretische Grundlagen und praktische Umsetzung

Springer Vieweg

Frank Witte
Landshut, Deutschland

ISBN 978-3-658-09963-3 ISBN 978-3-658-09964-0 (eBook)
DOI 10.1007/978-3-658-09964-0

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

Springer Vieweg

Gedruckt auf säurefreiem und chlorfrei gebleichtem Papier.

Springer Fachmedien Wiesbaden GmbH ist Teil der Fachverlagsgruppe Springer Science+Business Media (www.springer.com)

Vorwort

In diesem Buch gebe ich Ihnen Hinweise und konkrete Tipps, wie man beim Testmanagement und bei Softwaretests am besten vorgeht und werde dafür theoretische Grundlagen zusammen mit Anregungen zur Umsetzung in der betrieblichen Praxis verbinden. Ich verfüge über langjährige Erfahrung in diesem Bereich und habe in vielen Projekten in unterschiedlichen Branchen immer wieder gesehen, wie der Test der Software erfolgreich werden kann, wo es Fallstricke gibt und was getan werden muss, um sie zu vermeiden.

Zum Testen ist es immer erforderlich, das Testobjekt genau einzugrenzen – welche Anwendung, welches Betriebssystem, welche Testumgebung ... und die Eingrenzung des Untersuchungsgegenstands steht auch am Anfang dieses Buches.

Der Softwaretest findet oft in großen Projekten statt. Dabei gibt es jede Menge Störungen von außen: Profilierung einzelner Manager oder Mitarbeiter, Kampf zwischen Abteilungen, Managemententscheidungen, die einen gesamten Bereich betreffen und deren Sinn sich beim besten Willen nicht erschließen kann. Wenn es wichtiger ist, die Rendite des Unternehmens im Quartal um einen Zehntelprozentpunkt zu steigern und dadurch Projekte scheitern, wenn man meint, ein funktionierendes System durch nicht durchdachte Entscheidungen (etwa vorschnelle Offshore-Verlagerung, angebliche Synergieeffekte, bevorstehender Verkauf oder Merger des Unternehmens) außer Kraft setzen zu müssen und dabei ganze Teams restlos und nachhaltig demotiviert. Hin und wieder bekommt man solche Knüppel zwischen die Beine geworfen, so dass man sich nicht wundern muss, dass es dann manchmal keine nennenswerten Fortschritte gibt bzw. dass man sich im Gegenteil manchmal wundern muss, dass überhaupt noch ein Entwicklungsprojekt Fortschritte macht.

Dazu gehören auch Karrieredenken und Optimierung eigener Egoismen, sture Abgrenzungen, persönliche Eitelkeiten, Krisen und Dilettantismus, mangelnde Flexibilität, Bürokratismus, psychische Probleme wie Burnout, Boreout und innere Kündigung von Mitarbeitern, Machtansprüche und persönliche Eitelkeiten. Solche Rahmenbedingungen können jedes Projekt zum Scheitern bringen.

Manchmal werden laufende Projekte auch unerwartet gestoppt. Man wundert sich dann zwar warum man sparen will, koste es was es wolle, aber ändern kann man es in der Regel nicht. Oft erschließen sich die wahren Hintergründe dem Projektmitarbeiter dabei nicht.

Auch politische Termine, starre Budgetrestriktionen und mangelnde Kostentransparenz, kein Bewusstsein über die Folgen oder den Wunsch, die Wahrheit einfach nicht sehen zu wollen, belasten Innovationen weit mehr als es in der allgemeinen Wahrnehmung und selbst weit mehr als es in der gefilterten Wahrnehmung des Topmanagements gesehen wird.

Ein prominentes Beispiel für politische Termine ist die Einführung des LKW-Mautsystems Toll Collect:

Toll Collect wurde im März 2002 als Joint Venture der Deutschen Telekom, Daimler Chrysler und der französischen Cofiroute (Compagnie Financière et Industrielle des Autoroutes) gegründet. Cofiroute war einbezogen worden, weil von den Bewerbern Erfahrung mit vergleichbaren Projekten verlangt worden war und Telekom bzw. Daimler Chrysler dies nicht nachweisen konnten. Die Gesellschafteranteile von Toll Collect verteilten sich auf die Deutsche Telekom (45 %), Daimler Chrysler (45 %) und Cofiroute (10 %). Das Unternehmen beschäftigt nach eigenen Angaben rund 750 Mitarbeiter.

Mit dem System von Toll Collect sollte die Erfassung der Straßenbenutzungsgebühren für LKWs von einer pauschalen Gebühr auf eine wegstreckenerfasste Autobahngebühr umgestellt werden. Nach einigen politischen Auseinandersetzungen hatte Toll Collect im Juli 2002 den Zuschlag erhalten, im September 2002 wurde der Vertrag mit dem Bundesverkehrsministerium unterzeichnet. Für den Betrieb des Mautsystems sollte Toll Collect 12 Jahre lang ca. 650 Mio. Euro pro Jahr aus den Mauteinnahmen erhalten.

Der Staat verlangte schließlich von Toll Collect, der Tochterfirma des Konsortiums, 3,3 Milliarden Euro Schadenersatz plus 1,7 Milliarden Konventionalstrafe, weil das Mautsystem, das im Sommer 2003 eigentlich hätte in Betrieb gehen sollen, tatsächlich erst eineinhalb Jahre später funktionierte. 2005 erhob die damalige rot-grüne Regierung deswegen Schiedsklage. Toll Collect seinerseits klagte zurück, weil der Bund wegen der Start-Verzögerungen Geld einbehielt.

Bei der Planung von Toll Collect war man davon ausgegangen, dass bis Juni 2003 das System entwickelt wird und zum 1.9.2003 eingeführt wird, inklusive der Integrations- und Testphase sowie der Urlaubszeit.

Der alte Toll Collect Maut-Vertrag mit allen Anlagen und Nebenvereinbarungen hatten einen Umfang von 17.000 Seiten. Haben die Abgeordneten dieses Werk wirklich gelesen?

Des Weiteren meldete die EU Wettbewerbsbehörde wegen eines möglichen Monopols von Daimler Chrysler für die OBU Bedenken an. Die OBU ist die On Board Unit, also das Gerät was die Wegstrecke per Satellitenortungssystem GPS erfasst, mit der Anzahl der Fahrzeugachsen, der Schadstoffklasse, dem Fahrzeugkennzeichen und den gebührenpflichtigen Autobahnabschnitten zu einem Paket zusammenfasst und diese Daten mit einem Mobilfunksender per SMS an den Zentralrechner von Toll Collect schickt. Die EU Wettbewerbsbehörde verfügte dabei einige Auflagen für die behördliche Zulassung zur Bedingung, z. B. dass die OBU auch mit den Geräten anderer Hersteller kompatibel sein musste. Das führte zu einem erheblichen Terminverzug bei der Einführung des Systems, die für den 31.8.2003 angesetzt war. Dadurch wurde dem Staat ein erheblicher Einnahmeverlust zugefügt, weil das pauschale System zu diesem Zeitpunkt abgeschaltet wurde.

Es gab vor allem im Bereich der OBUs massive technische Probleme:

- OBUs reagierten nicht auf Eingaben.
- OBUs ließen sich nicht ausschalten.
- OBUs schalteten sich grundlos aus.
- OBUs zeigten unterschiedlich hohe Mautbeträge auf identischen Strecken an.
- OBUs wiesen mautpflichtige Autobahnstrecken als mautfrei aus.
- OBUs wiesen Strecken außerhalb des Autobahnnetzes als mautpflichtig aus.
- Siemens-Geräte passten nicht in den genormten Einbauschacht.

Diese Fehler wurden teilweise von der fehlerhaften Software verursacht, die die Geräte auch automatisch per GSM Mobilfunk aktualisieren sollte. Da die Fehler so gravierend waren, musste der Einführungstermin mehrfach verschoben werden bis schließlich der 1.1.2005 als Einführungstermin festgeschrieben wurde. Über die entstandenen Kosten, vor allem durch den Mautausfall, wurde lange kontrovers diskutiert: Die Bundesregierung hatte mit Mehreinnahmen von 2,6 Mrd. Euro im Vergleich zur alten Maut gerechnet und diesen Betrag bereits fest im Bundeshaushalt eingeplant.

Im März 2005 lief das System endlich mit einer Zuverlässigkeit von 99 %, aber die OBUs waren in ihrer Funktionalität eingeschränkt, da einige Fehler nicht so schnell beseitigt werden konnten. Vor allen Dingen die automatischen Updates der Software über GSM waren deaktiviert und konnten erst in einer nächsten Stufe zum 1.1.2006 eingeführt werden. Erst mit dieser Änderung konnten Streckenneubauten, Änderungen der Kilometerpauschale und Umwidmung von Bundesstraßen wegen der „Mautflucht“ vom System verarbeitet werden.

Dadurch erzielten das Projektmanagement und die verspätete Einführung einen Effekt auf die Umwelt, der weit über die Bedeutung des Mautsystems hinausging.

Zahlreiche Spediteure und Fuhrunternehmer kamen mit dem neuen System nicht klar und beklagten eine holprige Einführung. Das System verursachte auch erheblichen Mehraufwand bei LKW-Fahrern und in Logistikunternehmen. Das als Vorzeigeprojekt geplante System von Toll Collect hatte sich am Ende zu einem Lehrstück in Sachen Projektfehler entwickelt.

Es gibt also zahlreiche Kriterien, die Projekte verzögern, verteuern oder ruinieren, die außerhalb fachlicher Thematik liegen. Die meisten Projekte scheitern nicht wegen ingenieurstechnischer oder mathematischer Herausforderungen, nicht aufgrund komplizierter Technologie, sondern aufgrund mangelnder Kommunikation, trivialer Planungsfehler und relativ einfacher Probleme im betrieblichen Ablauf.

Es gibt genügend Experten und Bücher, die sich mit persönlichen Befindlichkeiten, nötigen Soft Skills und gruppendynamischen Prozessen intensiv befassen, so dass ich hierauf nicht eingehen werde.

Dieses Buch soll sich auf fachliche Themen beschränken, die mit dem Softwaretest direkt in Verbindung stehen.

Ein wesentlicher Grund, warum IT-Projekte in die Schieflage kommen, sind unklare Prozessvorgaben, schlechte Kommunikation und eine mangelhafte Organisation. Zumindest in diesem Bereich kann man mit professionellem Testmanagement einiges gegensteuern.

Ich konzentriere mich in der Betrachtung weitestgehend auf den Systemtest. Andere Testschritte und Projektphasen sind zwar zur Erreichung des Projektziels nicht weniger wichtig, sie sollen auch hier teilweise gestreift werden, wo es auf deren Vollständigkeit und Erfüllung ganz besonders ankommt, aber der Systemtest soll besonders beleuchtet werden.

Meine berufliche Erfahrung als Testkoordinator, Testmanager und Tester und die Projektarbeit als Freiberufler in den letzten 20 Jahren hat dieses Buch ebenfalls beeinflusst. Ähnlichkeiten mit lebenden oder toten Personen sind also rein zufällig – aber so manche Anekdote kann vielleicht plastisch das Spannungsfeld darstellen, in dem sich der Testverantwortliche befindet.

Wenn übrigens im Folgenden von Testmanagern, Testkoordinatoren, Testern … die Rede ist, meine ich immer auch Testmanagerinnen, Testkoordinatorinnen, Testerinnen und verwende nur aus Gründen der Lesbarkeit die männliche Form.

In diesem Buch soll das Thema Softwaretest demnach so betrachtet werden, als ob die Rahmenbedingungen stimmen, also man die Anwendung programmieren und fertig stellen möchte, dass das Entwicklungsprojekt überhaupt politisch gewollt ist, dass es keine gegenläufigen Tendenzen auf Managementebene gibt, man keine Querschläger von innen oder außen bekommt, es keine persönlichen Differenzen gibt und man sich nur der Sache widmen kann. Schon das ist alles bei Weitem nicht selbstverständlich, aber man muss bestimmte Fakten voraussetzen, um überhaupt in der Lage zu sein, eine Anwendung erfolgreich testen zu können.

Selbst mit durchweg positiven Rahmenbedingungen wird ein Testprojekt schon kompliziert genug. Softwaretest bezieht sich in der Regel auf komplexe, innovative Technologien, umfangreiche Systeme und verschachtelte Prozesse, die daher auch komplexe Testverfahren benötigen.

Inhaltsverzeichnis

1 Der Testprozess 1
- 1.1 Die Säulen im Testprozess 2
- 1.2 Phasen im Testprozess 3
- 1.3 Historie des Themas Softwaretest 4
- 1.4 Standardisierung von Testprozessen 5
- 1.5 Die Rolle des Testers im Testprozess 5
- 1.6 Rahmenbedingungen beim Softwaretest 7
- 1.7 Verfahren zur Prozessoptimierung 8

2 Grundsätze für Softwaretester 11
- 2.1 Testen zeigt die Anwesenheit von Fehlern 11
- 2.2 Vollständige Tests sind unmöglich 11
- 2.3 Mit dem Testen frühzeitig beginnen 12
- 2.4 Beim Testen flexibel sein 12
- 2.5 Tests müssen geplant werden 13
- 2.6 Testfälle regelmäßig prüfen und reviewen 13
- 2.7 Das Testumfeld berücksichtigen 14
- 2.8 Unterschiedliche Personen als Tester und Programmierer 14
- 2.9 Dokumentation und Nachvollziehbarkeit der Tests 14
- 2.10 Operation gelungen, Patient tot? 15
- 2.11 Ermittlung der Fehlerrate 15
- 2.12 Testnutzen 17
- 2.13 Testphilosophien 18

3 Testnormen 21
- 3.1 IEEE 829 Standard for Software Test Documentation 22
- 3.2 ISO-Standard 9126 22
- 3.3 ISO-Standard IEC-29119 25

4 Rollen und Verantwortlichkeiten im Testmanagement 27
4.1 Testmanager 27
4.2 Testkoordinatoren 28
4.3 Testdesigner 30
4.4 Testautomatisierer 30
4.5 Testsystemadministrator 31
4.6 Tester (Testdurchführer) 31
4.7 Qualifikation für Testverantwortliche 32

5 Grundlagen des Testmanagements 33
5.1 Integration der Testprozesse in den Life Cycle der Softwareentwicklung 34
5.2 Integrierte Toolunterstützung im gesamten Testprozess 34
5.3 Einheitliche Testumgebungen und Testdaten 35
5.4 Softwaretest als Management-Aufgabe 35
5.5 Industrialisierung und Standardisierung der Testprozesse 35

6 Marktsituation beim Softwaretest 37

7 Testvorbereitung 41
7.1 Teststrategie 41
7.2 Kennzahlen zur Wahl der geeigneten Teststrategie 44
7.3 Häufigkeit der Änderungen und Anzahl der Releasezyklen 46
7.4 Änderungsumfang pro Release 46
7.5 Testziele 47
7.6 Anforderungen an das Konfigurationsmanagement 49
7.7 Testendekriterien 50
7.8 Testorganisation 51

8 Requirements Engineering 53
8.1 Grundsätze des Anforderungsmanagements 53
8.2 Kriterien für die Formulierung von Requirements 55
8.3 Requirements Engineering in agilen Projekten 57
8.4 Beispiele zur Formulierung von Requirements 59
8.5 Nachträgliche Erstellung von Requirements 60
8.6 Probleme beim Requirements Engineering 61
8.7 Review der Requirements 63
8.8 Nebenabsprachen und Schattenprozesse 64
8.9 Verzögerungen beim Requirements Management 64

9 Teststufen 67
9.1 Modultest 67
9.2 Integrationstest 68
9.3 Systemtest 71

9.4 Abnahmetest . . . 73
9.5 Mischformen und Abgrenzung der einzelnen Teststufen . . . 74
9.6 Teststufen im V-Modell . . . 74
9.7 Blackbox- und Whitebox-Verfahren . . . 76
9.8 Mehrere Teststufen statt „Big Bang“ . . . 79

10 Testabdeckung und Überdeckungsmaße . . . 81

11 Fehlermanagement . . . 85
11.1 Definition von Fehlern . . . 86
11.2 Fehlerarten . . . 86
11.3 Kosten pro Fehler – die Barry-Boehm-Kurve . . . 87
11.4 Dynamik der Fehlerkosten . . . 89
11.5 Konsequenzen für die Fehlerbehandlung in den Teststufen . . . 93
11.6 Fehlerverfolgung . . . 94
11.7 Aufbau einer Fehlermeldung . . . 95
11.8 Tools für die Fehlerverwaltung . . . 96
11.9 Auswirkungen von Fehlern . . . 101
11.10 Schätzungen für die Fehlerdichte . . . 103
11.11 Programmierfehler – Beispiele . . . 104

12 Testplanung . . . 111
12.1 Erarbeitung der Teststrategie und der Testmaßnahmen . . . 111
12.2 Gemeinsames Verständnis über Testvorgehen und Testziele . . . 112
12.3 Tests nach Auslieferung des Produkts . . . 113
12.4 Testdaten . . . 113
12.5 Das Testprojekt im Entwicklungszyklus . . . 115
12.6 Zeitliche Planung der Testaktivitäten . . . 116

13 Testumgebung . . . 119

14 Testkonzeption . . . 123
14.1 Grundlagen . . . 123
14.2 Aufbau des Testkonzepts . . . 125
14.3 Testkonzept nach IEEE 829 . . . 128
14.4 Testarten, Testinhalte und Testphasen . . . 131
14.5 Testkonzept und betriebliche Realität . . . 132

15 Kennzahlen zur Bewertung von Tests . . . 135
15.1 Testprozessreife . . . 136
15.2 Ermittlung der Testproduktivität . . . 137
15.3 Kennzahlen für die Testeffektivität . . . 140
15.4 Die COCOMO-II-Gleichung . . . 140

15.5 Berechnung der Testdauer und des Testaufwands 142
15.6 Planungsrisiken bei der Aufwandsermittlung 143
15.7 Risikozuschläge in Testprojekten 145
15.8 Vom Entwicklungsaufwand zum Testaufwand 146
15.9 Vom Gesamtaufwand zur Detailbetrachtung 146
15.10 Berücksichtigung vorhandener Ressourcen 147
15.11 Genügend Fehlerpuffer miteinbeziehen 147
15.12 Erfahrungen aus früheren Projekten nutzen 148

16 Testvoraussetzungen 149

17 Beschreibung der Testfälle 151
17.1 Struktur der Testbeschreibung 151
17.2 Testspezifikation und Testimplementierung 152
17.3 Beschreibung der Geschäftsprozesse 153
17.4 Aufbau des einzelnen Testfalls 153
17.5 Strukturierung der Testfälle 155
17.6 Anzahl Testfälle 156
17.7 Abhängigkeiten zwischen Testfällen 157
17.8 Priorisierung von Testfällen 158
17.9 Funktionale und nichtfunktionale Testfälle 158

18 Testmethoden 165
18.1 Äquivalenzklassenbildung 165
18.2 Unterschiedliche Testtiefen 169
18.3 Überdeckungsmaße 170
18.4 Test Maturity Model 177
18.5 Test Process Improvement (TPI) 177

19 Testdurchführung 179
19.1 Änderungen der Testfallbeschreibung während der Testdurchführung 180
19.2 Strukturiertes Testen und exploratives Testen 180
19.3 Intuitive Testfallermittlung 182
19.4 Durchführung explorativer Tests 183

20 Der Testbericht 185
20.1 Kriterien für Testberichte 185
20.2 Beschreibung der Rahmenbedingungen 186
20.3 Bericht der Testergebnisse 187

21 Produktiveinführung von Systemen . . . 191
21.1 Pilotbetriebe . . . 192
21.2 Produktivnahme mit Übergangsphase . . . 193
21.3 Hilfsmodelle . . . 194

22 Reviews im Testprozess . . . 197
22.1 Allgemeines über Reviews . . . 197
22.2 Vorteile von Reviews . . . 197
22.3 Reviewprozess . . . 199

23 Werkzeuge für die Unterstützung des Testmanagements . . . 201
23.1 Evaluation von Testtools . . . 201
23.2 Suche geeigneter Werkzeuge . . . 202
23.3 Anforderungen an geeignete Test-Tools . . . 204

24 Die optimale Testabdeckung . . . 207

25 Testmetriken . . . 209
25.1 Arten von Testmetriken . . . 210
25.2 Bewertung der Komplexität von Testfällen . . . 211
25.3 Problematik von Metriken . . . 215
25.4 Verwässerung von Metriken . . . 216
25.5 Metriken, bei denen man (fast) nur verlieren kann . . . 218
25.6 Aussagen unterschiedlicher „Key Performance Indicators“ . . . 219
25.7 Beispiel für eine Fehlermetrik . . . 222

26 Testautomatisierung . . . 223
26.1 Chancen und Risiken der Testautomatisierung . . . 224
26.2 Vorgehen bei der Testautomatisierung . . . 225
26.3 Planung der Testautomatisierung . . . 226
26.4 Voraussetzungen für automatische Testdurchführung . . . 227
26.5 Codenahe und GUI-Testautomatisierung . . . 227
26.6 Untersuchungen vor der Automatisierung . . . 228
26.7 Ablauf der Testautomatisierung . . . 228
26.8 Testergebnisse bei automatisierten Tests . . . 229
26.9 Qualifikation der Softwaretester bei automatisierten Tests . . . 230
26.10 Automatische Testskripts und Änderungen in der Software . . . 231
26.11 Automatische Generierung von Testdaten . . . 232
26.12 Fehlerfreie Ergebnisse ergeben vollständige Reproduzierbarkeit . . . 232
26.13 Fazit . . . 233

27 Offshoring von Tests . . . 235

28 Test von Internet-Anwendungen . . . 241
28.1 Struktur bei Web-Tests . . . 241
28.2 Fragen an die Web-Applikation . . . 242
28.3 Phasen beim Web-Test . . . 244
28.4 Testabdeckung beim Test von Internetanwendungen . . . 245
28.5 Test unterschiedlicher Sprachen . . . 246
28.6 Look and Feel . . . 247
28.7 Test nach Zielgruppen . . . 247
28.8 A/B-Testing und Multivariate Testing . . . 248

29 Testen in der Cloud . . . 255

30 Mobile Testing . . . 257
30.1 Besonderheiten beim Mobile Testing . . . 257
30.2 Auswirkungen auf das Testmanagement . . . 258

31 Agile Testing . . . 259
31.1 Besonderheiten beim Agile Testing . . . 259
31.2 Testmanagement in agilen Testprojekten . . . 261
31.3 Kombination traditioneller Testmethoden und agiler Methoden . . . 262

Testen ist kein Allheilmittel . . . 263

Fazit . . . 265

Literatur . . . 267

Sachverzeichnis . . . 271

Der Testprozess

1

Der Softwaresteht besteht aus mehreren Phasen. Ein definierter effizienter Testprozess verbessert die Abläufe und erhöht die Softwarequalität.

Der idealtypische Testprozess kann wie in Abb. 1.1 dargestellt werden.

Das Testmanagement erstreckt sich von der Testplanung über die Erstellung der Testspezifikation und Durchführung der Tests bis hin zur Protokollierung der Testergebnisse und der Auswertung der Tests.

Das Testmanagement organisiert, koordiniert und überwacht die entsprechenden Aktivitäten durch:

- Incident Management: organisatorischer und technischer Prozess der Reaktion auf erkannte oder vermutete Störungen in IT-Bereichen sowie hierzu vorbereitende Maßnahmen und Abläufe.
- Problem Management: ermitteln, bewerten und Korrektur aufgetretener Fehler.
- Change Management: Bearbeitung zusätzlicher oder geänderter Anforderungen während des Projektablaufs.
- Release Management: Planung und Konfiguration der einzelnen Software Releases.

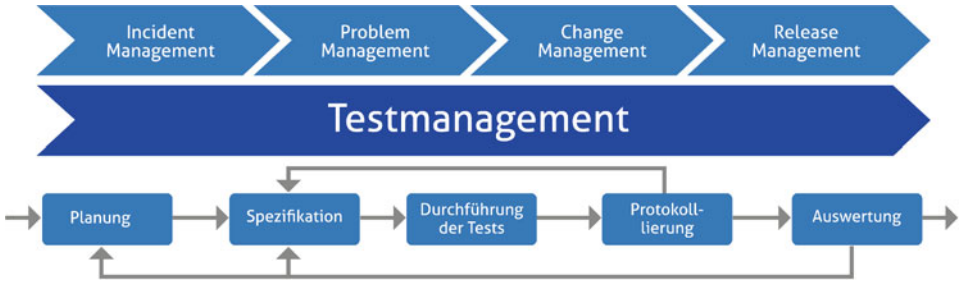

Abb. 1.1 Der Testprozess

F. Witte, *Testmanagement und Softwaretest*, DOI 10.1007/978-3-658-09964-0_1

Ein definierter Testprozess dient dazu, die einzelnen Testaktivitäten voneinander abzugrenzen, Ablaufpläne zu verfeinern und inhaltliche und zeitliche Abhängigkeiten der einzelnen Testaktivitäten aufzuzeigen.

Der weltweit verbreitete ISTQB-Testprozess unterscheidet folgende Aktivitäten:

- Planung und Steuerung,
- Analyse und Design,
- Realisierung und Durchführung,
- Auswertung und Bericht,
- Abschluss.

Die einzelnen Aktivitäten des Testprozesses sind nicht strikt nacheinander abzuarbeiten, eine zeitliche Überlappung wird sich in der Praxis weder vermeiden lassen noch ist sie unbedingt erforderlich.

1.1 Die Säulen im Testprozess

Ein erfolgreicher Testprozess steht auf mehreren Säulen, die in diesem Buch näher beschrieben werden (Abb. 1.2).

Das Testmanagement muss bereits von Anfang an in die Softwareentwicklung mit eingebunden werden. Die Teststrategie definiert die grundlegenden Festlegungen, ein methodisches Vorgehen bei der Organisation der Testprozesse legt die Basis für einen erfolgreichen Ablauf. Eine Testplanung und Konzeption des Testvorgehens definiert wer, was, wann, wie testet und welche Umgebungsparameter erforderlich sind. Dazu benötigt man fachliches Know-how über die Anwendung, die betrieblichen Abläufe und standardisierte Vorgehensweisen. Professionelle Testtools unterstützen die administrative Abwicklung.

Abb. 1.2 Säulen im Testprozess

1.2 Phasen im Testprozess

Der grundsätzliche Testprozess kann aus dem V-Modell mit folgenden Stufen abgeleitet werden:

- Testplanung,
- Testdesign,
- Testspezifikation,
- Testdurchführung,
- Testprotokollierung,
- Testauswertung.

In der Phase der Testplanung werden das Testkonzept und der Testplan erstellt. Dabei werden das Testobjekt, die Testumgebung, die Konfiguration des Testsystems und die Testressourcen beschrieben. Dabei wird festgelegt, welcher Testumfang mit welchen Tools getestet werden soll.

Beim Testdesign wird die Testplanung verfeinert. Die Testanforderungen, nötigen Testszenarien und Testendekriterien werden in diesem Schritt dokumentiert. Je nach Umfang des Projekts kann diese Phase aber auch mit der Testplanung zusammenfallen.

In der Phase der Testspezifikation werden die Testfälle im Detail beschrieben. Die Testvoraussetzungen, die erforderlichen Eingaben und erwarteten Ausgaben werden dokumentiert und Abhängigkeiten zwischen den einzelnen Testfällen ermittelt. Dabei werden sowohl funktionale als auch nichtfunktionale Anforderungen berücksichtigt.

Die Testdurchführung geschieht manuell oder automatisiert, in der Praxis sind meist Mischformen anzutreffen. Die Testdurchführung ist fast immer iterativ, weil die Software phasenweise entwickelt wird, die entwickelten Features demnach im Laufe der Zeit erweitert werden und weil behobene Fehler immer wieder mit Regressionstests nachgetestet werden müssen. Die Entwicklung schreitet in der Regel auch in Phasen voran, so dass der Testumfang sich während der Projektdauer um neue Testaktivitäten erweitert.

Bei der Testprotokollierung werden die Testergebnisse aufgezeichnet, manuell mit Checklisten oder Testberichten oder automatisiert mittels Logfiles. Diese Ergebnisse werden im Rahmen der Testauswertung analysiert und aufbereitet. Dabei stellt man evtl. fest, dass man zu früheren Phasen im Testprozess zurückkehren muss, etwa wenn die Testspezifikation fehlerhaft oder lückenhaft war.

Dieser Testprozess soll aber nur einen grober Rahmen zur Orientierung geben, und keine starre Vorgabe sein. In agilen Testansätzen geht man verstärkt dazu über, den Testprozess möglichst offen zu halten und den Umständen der jeweiligen Aufgabenstellung anzupassen.

Modellhafte Ansätze sind immer abzuwandeln. Jede Aufgabenstellung, jede Software, jedes Testprojekt und jedes Team steht vor individuellen Herausforderungen, wie es die Phasen effizient umsetzt.

1.3 Historie des Themas Softwaretest

Softwaretests waren bis ca. 1975 nahezu unbekannt. Ende der 70er Jahre wurde das erste kommerzielle Testlabor eröffnet, da erste Unternehmen die Notwendigkeit eines Softwaretests erkannten, aber dafür keine Ressourcen zur Verfügung hatten, und die Testtheorie erst ansatzweise begonnen hatte. Als sich das Thema Softwaretest in den 80er und 90er Jahren des letzten Jahrhunderts langsam etablierte, kam es des Öfteren vor, dass die Software fast fertiggestellt war und erst dann getestet wurde. Eigene Experten oder eine definierte Testabteilung waren bei vielen Unternehmen noch nicht vorhanden. Ein Berufsbild für den Softwaretester war damals erst im Entstehen, eine Standardisierung von Testprozessen nur ansatzweise vorhanden.

Nach mehreren negativen Erfahrungen und einigem Lehrgeld ist in den Unternehmen auf breiter Front das Bewusstsein gewachsen, wie wichtig ein entwicklungsbegleitender Softwaretest ist, um den Ansprüchen an qualitativ hochwertige Lieferungen zu genügen – sei es, dass die Software an Kunden geliefert wird, oder dass es sich um interne Anwender handelt (z. B. ein Warenwirtschaftssystem das in der eigenen kaufmännischen Abteilung zur Fakturierung eingesetzt werden soll).

Bereits 1979 formulierte G. Myers Kriterien für den Softwaretest, die noch heute ihre Gültigkeit besitzen:

1. Ein notwendiger Bestandteil eines Testfalls ist die Definition der erwarteten Werte oder des Resultats.
2. Ein Programmierer versucht nicht, sein eigenes Programm zu testen.
3. Die Resultate eines jeden Tests sollen gründlich überprüft werden.
4. Testfälle müssen für ungültige und unerwartete ebenso wie für gültige und erwartete Eingabedaten definiert werden.
5. Ein Programm darauf hin zu untersuchen, ob es nicht tut, was es tun sollte, ist nur die eine Hälfte der Schlacht. Die andere Hälfte besteht darin zu untersuchen, ob das Programm etwas tut, was es nicht tun soll.
6. Wegwerftestfälle sollten vermieden werden, es sei denn, das Programm ist wirklich ein Wegwerfprogramm.
7. Es sollten keine Testverfahren unter der stillschweigenden Annahme geplant werden, dass keine Fehler gefunden werden.
8. Testen ist eine extrem kreative und intellektuell herausfordernde Aufgabe.
9. Testen ist der Prozess, ein Programm in der Absicht auszuführen, Fehler zu finden.
10. Ein guter Testfall ist dadurch gekennzeichnet, dass er mit hoher Wahrscheinlichkeit einen bisher unbekannten Fehler anzuzeigen imstande ist.
11. Ein erfolgreicher Testfall ist dadurch gekennzeichnet, dass er einen bisher unbekannten Fehler entdeckt.

1.4 Standardisierung von Testprozessen

Eine der Hauptursachen für die Fehleinschätzung von Aufwänden und Problembereichen ist, dass die Softwareindustrie wesentlich jünger ist als andere Industrien. Kaum mehr als 40 Jahre sind vergangen, seit die ersten Softwareunternehmen entstanden. Das sollte man sich immer wieder vor Augen führen, wenn man über ihre künftige Entwicklung spricht. Das heißt: Die Softwareindustrie kommt erst allmählich in die „reife" Entwicklungsphase, in der die klassischen Industriezweige – wie zum Beispiel die Elektro-, die Automobilindustrie oder der Maschinenbau – sich schon lange befinden. Wenn man die heutige Reife der Softwareindustrie mit der klassischer Industrien vergleicht, befindet sich die Softwareindustrie in etwa im Jahre 1920 oder 1930.

Die Automobilindustrie treibt die Standardisierung in der Produktion seit Henry Ford voran. Die Pharmaindustrie setzt schon lange auf Cluster und Kooperationen mit Mitbewerbern in Entwicklung und Marktbearbeitung. Für die Elektro- oder Textilbranche ist die Auslagerung der Produktion und Entwicklung in Offshore-Länder eine Existenzbedingung. All diese Trends haben inzwischen auch die Softwareindustrie erreicht.

Deswegen müssen Software-Hersteller ihre Geschäftsmodelle in immer kürzeren Zeitabständen überdenken. Sie müssen regelmäßig ihre Geschäftsprozesse hinterfragen und untersuchen, wie sie höhere Qualität und geringere Kosten miteinander vereinbaren können. Unternehmen müssen über ihre Fertigungstiefe nachdenken und Tätigkeiten auslagern oder automatisieren. Außerdem müssen das Management professionalisiert und die internen Prozesse neu überdacht werden.

Software hat inzwischen alle unsere Lebensbereiche durchdrungen: sei es im Auto oder im Haushalt durch embedded Systeme, in der Fabrik, im Büro, oder in einem Online Shop. In den App Stores werden bereits heute über 2 Millionen Apps zum Download angeboten, und die Zahl steigt ständig weiter. Da jede Software vor Auslieferung an den Endnutzer bzw. Auftraggeber getestet werden muss, ist es wichtig zu prüfen, wie diese Herausforderungen unter Berücksichtigung der Kostenrahmen und der Qualitätsziele umgesetzt werden können.

1.5 Die Rolle des Testers im Testprozess

Parallel zur Professionalisierung der Softwareindustrie hat sich das Rollenbild des Testers in den vergangenen Jahren massiv verändert. Noch vor etwa zwanzig Jahren war es in den meisten Unternehmen nicht einmal existent. Die in den primär sequentiellen Vorgehensmodellen definierte Phase „Test" wurde oft als Knautschzone, als Puffer für die in der Regel zu enge Projektplanung angesehen. Der Aufwand für den Softwaretest und somit auch für die Tester war schon im Vorfeld als Einsparungspotenzial überzogener Projektbudgets definiert. Man wollte nicht viel in eigene Tester investieren, sondern die Entwickler selber testen lassen („der Auftraggeber sieht sie sich ja selbst noch mal an") und Tester galten oft als minderbegabte Entwickler. Die Sicht auf die Bedeutung des

Softwaretests war in etwa so, als ob man vor der Einweihung einer neuen Brücke gerade einmal einen einzigen LKW über die Brücke fahren lässt. Hält die Brücke, wird sie freigegeben, stürzt sie am nächsten Tag unter der Last der Autos ein, war der Lastkraftwagenfahrer schuld.

Dass ein Entwickler die selbst erstellte Software testet, widerspricht völlig der Philosophie, ein Produkt herzustellen. „Testing is a destructive process, even a sadistic process" (Myers). Derjenige, der ein Produkt definiert, entwickelt und implementiert, ist am wenigsten dazu geeignet, die Ergebnisse seiner Tätigkeit destruktiv zu betrachten. Er wird schon unbewusst das Interesse daran haben, nur das Positive an seinem Erzeugnis zu sehen.

Daher soll nie der Entwickler das Endprodukt testen. Diese Erkenntnis hat sich zum Glück (nach jahrelanger Überzeugungsarbeit und vielen Fehlversuchen) inzwischen in den Unternehmen durchgesetzt. Man bezweifelt heute nicht mehr, dass es für den Systemtest eine eigene Instanz geben muss.

Mit jedem neuen Schaden wuchs die Erkenntnis, den Softwaretest ernster zu nehmen, besser zu planen und professioneller umzusetzen. Der Softwaretest wurde im Lauf der Jahre zu einer eigenen Disziplin und zu einer unabhängigen Instanz und einem selbstbewussten Rollenverständnis. Durch standardisierte Ausbildungen und Zertifikate, z. B. vom ISTQB, hat sich das Berufsbild des Softwaretesters in den letzten Jahren klarer geformt. Testen wird nicht mehr als eine Begleiterscheinung der Programmierung, sondern als eigenständige Disziplin betrachtet. Das Bewusstsein, dass der Softwaretest kein Studentenjob, keine Nebenbeschäftigung für nicht ausgelastete Sachbearbeiter oder gescheiterte Programmierer sein darf, ist inzwischen auf breiter Front vorhanden. Mit dieser Professionalisierung kamen auch Testtechniken, Testmethoden und Testwerkzeuge auf, und die Frage, ob und wozu überhaupt getestet werden soll, wird nicht mehr gestellt.

Die fortschreitende Digitalisierung der Umwelt macht einen zunehmenden Einsatz von Softwareprodukten erforderlich, die ständig weiterentwickelt und an neue Strukturen angepasst werden müssen. Daher bietet sich für den Softwaretester ein immer breiteres Betätigungsfeld. Vor allem größere Unternehmen der IT-Branche mit eigener Software-Entwicklungsabteilung verfügen oftmals über ein eigenes Team von Softwaretestern, welches die Produkte ausgiebig testet, ehe sie letztendlich auf den Markt kommen. Aber auch für Existenzgründer ist die Tätigkeit als Softwaretester eine lohnenswerte Aufgabe.

Umso wichtiger wird aber die Frage, wie man Testprozesse optimieren und effizient umsetzen kann, damit der Softwaretest mit der zunehmenden Professionalisierung und zunehmenden Reife der Softwareindustrie angemessen Schritt halten kann. Da das Thema Softwaretest mit der steigenden Bedeutung auch zunehmend Kosten verursacht, muss man schon im kaufmännischen Sinne bestrebt sein, Testprozesse zu verbessern, um die Softwarequalität zu erhöhen ohne die Kostenrahmen der Projekte zu sprengen.

1.6 Rahmenbedingungen beim Softwaretest

Die Professionalisierung der Software-Entwicklung führte dazu, dass auch dem Softwaretest mehr Bedeutung zugewendet werden muss:

- komplexere Produkte: Dieser Trend wird bei Autos oder elektronischen Produkten besonders transparent: Das Endprodukt beinhaltet mehr Features, unterschiedliche Komponenten, die in Wechselwirkungen zueinander stehen. Eine Änderung an irgendeiner Stelle im System führt zu einem Fehlverhalten an einer ganz anderen Stelle, teilweise völlig unerwartet und nicht vorhersehbar. Das Navigationsgerät im Auto muss unabhängig vom MP3-Player und der Klimaanlage funktionieren.
- schnellere Produktzyklen: Der Erwartungsdruck, neue Produkte auf den Markt zu bringen, die Dynamik im Markt ist gestiegen. Durch das Internet werden Produkte vergleichbarer, die Informationsfülle ist gewachsen, die Datenfülle angestiegen. Dadurch werden Innovationen und Neuentwicklungen beschleunigt, das menschliche Wissen wächst in immer kürzeren Intervallen. Das Internet erlaubt mehr Vergleichbarkeit, die Konkurrenz ist nur noch einen Mausklick vom eigenen Produkt entfernt. Die Reifephase ist verkürzt, schon aus Gründen kürzerer Konjunkturzyklen und der Erwartung schnellerer Return-on-Invests. Gerade für Hochlohnländer, die nicht über den Preis konkurrieren können, ist dabei höhere Komplexität und bessere Qualität zu einem wesentlichen Unterscheidungsmerkmal geworden. Die schnelle Reaktion auf geänderte gesetzliche Richtlinien, Marktgegebenheiten und Kundenanforderungen bedeutet, dass Unternehmen bessere Software in kürzeren Abständen ausliefern müssen. Erschwerend kommt hinzu, dass sich Änderungen vielfach noch während des Entwicklungsprozesses ergeben und entsprechende erste Anpassungen bereits vor Fertigstellung der Software durchzuführen sind.
- stärkerer Wettbewerb: Die Globalisierung macht Produkte und Dienstleistungen leichter vergleichbar und austauschbarer. Durch eine international wachsende Zahl von Unternehmen und Arbeitskräften nimmt der Innovationszyklus zu. Eine größere Freiheit von Kapitalströmen, mehr Exporte bis zu einer stärkeren Mobilität von Arbeitskräften bewirken eine stärkere Internationalisierung und eine Erhöhung des Wettbewerbs.
- verteilte Entwicklung: Komplexe Produkte bestehen aus zahlreichen Komponenten, die aufeinander abgestimmt sein müssen. Die Fertigungstiefe hat in allen Industrien abgenommen, die Spezialisierung steigt, man muss Komponenten von anderen Entwicklungsgruppen, anderen Firmen oder anderen Ländern im eigenen Produkt mit integrieren. Die Entwicklungstiefe hat ebenfalls abgenommen, der Lieferant ist nicht mehr nur eine verlängerte Werkbank, sondern spezialisiert sich auf die Optimierung seiner Komponenten. In der Automobilindustrie wird das besonders deutlich. Durch die gestiegenen Anforderungen an die Software-Integration steigen auch die Anforderungen an den Softwaretest.

1.7 Verfahren zur Prozessoptimierung

Um Softwareentwicklungsprozesse zu optimieren und vergleichbar beurteilen zu können, gibt es inzwischen zahlreiche Verfahren (CMMI, SPICE usw.). Diese Verfahren können Hinweise geben, müssen aber konsequent umgesetzt werden, um die entsprechenden Wirkungsgrade zu entfalten.

Oft werden Normen nur übergestülpt und stehen als theoretischer Block neben den eigentlichen Geschäftsprozessen. Jeder noch so gute Ansatz muss aber adaptiert, abgeändert und verfeinert werden, sonst bleibt das neue Verfahren eine Worthülse und lässt frustrierte Entwickler und Tester zurück. Dazu ist es wichtig, den Nutzen und die wesentlichen Inhalte eines Verfahrens zu verstehen.

Eine Beurteilung der Softwareproduktreife an sich findet aber meist gar nicht statt. Selbst eine optimale Gestaltung der Entwicklungsprozesse reicht nicht aus, um auch eine hohe Produktqualität zu garantieren. Vor allem die Bedeutung der Testphase wird dabei leicht unterschätzt.

Es leuchtet eher ein, mit zusätzlichen Entwicklern weitere Features zu programmieren, weil es dabei um Herstellung geht, weil die Produktivität im Vordergrund der Betrachtung steht, weil klar wird, dass man schneller fertig wird oder ein umfangreicheres Produkt mit zusätzlichen Extras bieten kann. Man sieht dann vor seinem inneren Auge ein umfassenderes, kompliziertes und innovativeres Produkt. Wenn man dieselbe Kapazität, um die man die Entwicklung aufstocken könnte, aber in den Test steckt, sieht man auf den ersten Blick weder neue Features noch zusätzliche Innovation. Im Gegenteil, je mehr man testet, desto mehr steigen die Kosten und desto mehr verschiebt sich der Fertigstellungstermin nach hinten. Qualität versteckt sich manchmal hinter der glänzenden Oberfläche, und nicht immer hat der Softwaretest eine kräftige Lobby im Unternehmen. Das liegt teilweise auch an der Qualität des Testmanagements.

Softwaretestern fehlt es häufig an Eigenmarketing. Ein erfolgreicher Softwaretester benötigt vor allem ein gutes Gedächtnis, Verständnis für Zahlen, Detailgenauigkeit, Akribie und ein logisches Denkvermögen. SAP plant, dass bis zum Jahr 2020 ca. 1 % seiner Belegschaft Autisten sein sollen, die vor allem als Softwaretester eingesetzt werden sollen. Ein Schweizer Dienstleister für Softwaretest, Asperger Informatik, stellt bewusst Mitarbeiter mit dem Asperger-Syndrom, einer leichteren Form von Autismus, ein. Diese Mitarbeiter verfügen über herausragende Qualitäten für Konzeption, Durchführung und Analyse von Softwaretests aber nicht unbedingt über Soft Skills und haben manchmal Defizite in der zwischenmenschlichen Kommunikation. Häufig geht es bei Testberichten um komplexe Zusammenhänge, endlose Zahlenreihen und Detailprobleme, die nur schwer in eindrucksvollen bunten Bildern zu präsentieren sind.

Wenn die Testabteilung nur als notwendiges Übel angesehen wird, sie nur über das Image der unbequemen Erbsenzähler verfügt und nur als Sammelbecken gescheiterter Programmierer gilt, die für die kreativen Aufgaben unfähig sind und nur vorgegebene Checklisten ankreuzen, dann wird sie im Unternehmen nicht das Ansehen bekommen das ihr zusteht und das sie dringend benötigt. Es ist unbedingt erforderlich, dass der Entwick-

lungsleiter und der Leiter der Testabteilung auf Augenhöhe miteinander verhandeln. Bei Projektmeetings muss der Leiter der Testabteilung mit am Tisch sitzen und eigene Akzente setzen. Der entsprechenden Stellung des Testers muss von der Betriebsorganisation Rechnung getragen werden.

Dem Leiter der Testabteilung kommt sogar in besonderer Weise die Aufgabe zu, seinen Posten auch im Networking auszufüllen, gerade wenn seine Abteilung unter den bezeichneten Imageproblemen leiden sollte.

Grundsätze für Softwaretester

2

Es gibt einige generelle Leitlinien für Softwaretester, eine Testphilosophie die allen Softwaretest-Aktivitäten zugrunde liegt und als gemeinsame Richtschnur in jedem Testprojekt, ob groß oder klein, vorhanden sein sollte. Diese Grundsätze verhelfen zu einem gemeinsamen Verständnis und einer richtigen Einordnung der Testaktivitäten.

2.1 Testen zeigt die Anwesenheit von Fehlern

Testaktivitäten schlagen fehl und erzeugen Fehler. Das bedeutet im Umkehrschluss: Je höher die Testabdeckung ist, desto geringer ist das Risiko, dass noch unentdeckte Fehler im Testobjekt vorhanden sind. Testen zeigt aber nicht, dass die Software fehlerfrei ist, selbst wenn keine Fehler nachgewiesen werden. Testen ist kein mathematischer Beweis.

Hin und wieder wird der Tester angegriffen, wenn ein Fehler während des Tests doch nicht gefunden wurde, sondern erst nach dem Rollout. In diesem Fall ist es wichtig, das Nichtentdecken des Fehlers zu analysieren, aber auch darauf hinzuweisen, dass die Testabdeckung bewusst begrenzt war. Vielleicht wurde im Moment der Testplanung ein Geschäftsvorfall für nicht relevant genug erachtet, um im Test nachgewiesen zu werden, und/oder das vorhandene Budget reichte nicht aus, die Testabdeckung entsprechend zu erweitern. Beim Softwaretest ist es wichtig, aus den begrenzten Mitteln so viel Qualität wie möglich zu erreichen, aber keine Testabdeckung von 100 %.

2.2 Vollständige Tests sind unmöglich

Softwaretests beziehen sich immer auf die Betrachtung von Stichproben, es werden nicht alle möglichen Eingabeparameter und deren Kombinationen mit allen möglichen Vorbedingungen getestet, sondern diejenigen, die am ehesten praxisrelevant sind. Der Testaufwand richtet sich dabei nach Priorität und Risiko. Der Testaufwand ist dabei immer mit

F. Witte, *Testmanagement und Softwaretest*, DOI 10.1007/978-3-658-09964-0_2

dem Testnutzen abzugleichen. Wo es sinnvoll erscheint, sollten Tests automatisiert werden um die Testabdeckung zu steigern. Tester werden oft dafür angegriffen, dass sie etwas nicht bemerkt haben, was dann beim Kunden oder beim Endanwender festgestellt wurde. Der Tester muss in diesem Fall begründen können, warum dieser Test nicht durchgeführt wurde und warum er nicht vorgesehen war. Der Tester sollte das aber nicht als persönliche Schuldzuweisung verstehen. Teilweise wird manchmal ein verheerender Umkehrschluss gezogen: Wozu leisten wir uns im Unternehmen eine teure Testabteilung, wenn sie die Fehler trotzdem nicht entdeckt? Ohne Tester hätten wir dasselbe (desolate) Ergebnis gehabt, aber nicht so viel Geld dafür bezahlen müssen. In diesem Fall empfiehlt sich immer die Argumentation, dass ohne Softwaretest die Kosten noch wesentlich höher und das Desaster noch wesentlich größer gewesen wäre, weil dann auch andere Fehler nicht aufgefallen wären.

Ein Test dient zur Feststellung, ob ein Programm wie in der Spezifikation festgehalten funktioniert und nichts anderes tut. Ein Test zeigt aber nicht, ob die Spezifikation wirklich sinnvoll ist. Man kann also erfolgreich getestet haben und trotzdem ein Produkt entwickelt haben, das an den Vorstellungen und Wünschen der Anwender vorbeigeht.

2.3 Mit dem Testen frühzeitig beginnen

Mit dem Test sollte so früh wie möglich im Software- bzw. Systemlebenszyklus begonnen werden. Selbst wenn noch keine testfähige Software zur Verfügung steht, kann man bereits die Requirements analysieren, das Testkonzept schreiben, Testprozeduren dokumentieren und die nötigen Umweltvoraussetzungen für den Test schaffen damit man gut vorbereitet ist, wenn die erste lauffähige Version zur Verfügung steht. Systematisches Testmanagement begleitet die Testaktivitäten von Anfang an und in jeder Entwicklungsstufe. Dadurch lassen sich Fehler bereits in den Entwicklungsstufen aufdecken, in denen sie entstanden sind und werden nicht erst in späteren Phasen gefunden. Wenn Fehler frühzeitig im Lebenszyklus des Projekts gefunden wurden, verkürzt das Korrekturzyklen und vermindert Integrationsaufwand. Grundsätzlich ist es vorteilhaft, wenn so viele Vorbereitungsaktivitäten wie möglich schon vor dem Test getroffen wurden. Andererseits sieht man manche Features erst wenn man die Software erst tatsächlich einem Test unterzogen hat, auch hier gibt es also eine Grenze. Oft wird aber gerade in der Phase, in der man noch keine testfähige Anwendung zur Verfügung hat, wertvolle Zeit versäumt und in einer späteren Phase kurz vor Lieferung des Endprodukts wird es dann umso hektischer im Team.

2.4 Beim Testen flexibel sein

Wenn in einem bestimmten Softwaremodul ein Fehler gefunden wurde, sind erfahrungsgemäß weitere Fehler nicht weit. Fehler treten in der Regel nicht gleichmäßig verteilt über alle Komponenten auf, viel wahrscheinlicher ist die Häufung von Fehlern in einzel-

nen Komponenten. Für das Testen heißt das, dass flexibel auf solche erkannten Häufungen eingegangen werden muss. So wichtig eine exakte Testplanung ist, so wichtig ist es darauf zu achten, kreativ mit neuen Herausforderungen umzugehen. Das zeigt sich vor allem in explorativen Tests, also Tests die durch Ausprobieren von Fehlerfällen oder neuen, nicht vorgesehenen Kombinationen, durchgeführt werden. Tester müssen eine gewisse Phantasie entwickeln können und sich auch in den Nutzer hineinversetzen, der das System nicht streng nach der vorgegebenen Anweisung benutzt. Der Tester soll gerade auch die Funktionen genau untersuchen, die nicht exakt spezifiziert sind.

2.5 Tests müssen geplant werden

Um zu verlässlichen Aussagen zu Kosten und Terminen zu kommen, sind die Testfälle aufgrund der Komplexität der Anwendung zu bewerten und daraus Tests abzuleiten. Dabei muss man die Anzahl der voraussichtlich benötigten Testzyklen, den Umfang der Testbeschreibung und die Dauer für die Durchführung der Tests berücksichtigen und mit den vorhandenen Ressourcen abstimmen. Gerade dann wenn es mehrere voneinander abhängige Projekte im Unternehmen gibt, ist es wichtig, eine verlässliche Testplanung zu haben und sie jederzeit bei neuen Erkenntnisse zu korrigieren und zu aktualisieren.

2.6 Testfälle regelmäßig prüfen und reviewen

Neue Anforderungen, geänderte Umgebungsvariablen und erweiterte Szenarien erfordern neue Testfälle. Beim Testen darf man also nicht nur buchstabengetreu der vorhandenen Testspezifikation folgen, sondern sie kritisch betrachten, sie ergänzen und aktualisieren wo es vonnöten ist. Unerfahrene Tester und eine fehlende Testmethodik führen zu redundanten und fehlenden Testfällen. Mehrfache Tests (evtl. in verschiedenen Testphasen) für dieselbe Funktion führen zu Kostensteigerung und Terminverzug. Fehlende Testfälle führen zu mangelnder Testabdeckung und damit zu einem erhöhten Risiko fehlerhafter Software und geringerer Qualität des Endprodukts. Gerade erfahrene Tester müssen dazu angehalten werden, ihre Testfälle exakt zu dokumentieren, damit das Know-how und die Kreativität mit denen mittels explorativer Testfälle zusätzliche Konstellationen oder Seiteneffekte nachgewiesen wurde auch für künftige Versionen und Projekte produktiv genutzt werden kann. Ein strukturiertes Testmanagement und eine methodische Planung von Testaktivitäten sind entscheidend für den Testerfolg. Aus diesem Grund ist es auch so wichtig, die einzelnen Testphasen voneinander abzugrenzen und sich im Vorfeld genau zu überlegen, welche Funktionalität an welcher Stelle mit welcher Testumgebung sinnvollerweise zu testen ist. Dabei ist es auch entscheidend, immer wieder die Testfälle kritischen Reviews zu unterziehen. Reviews mögen anfangs als Kostentreiber erscheinen, aber sie sind wichtig für den Erfolg des Endprodukts und dienen dadurch in Summe sogar zur Einsparung von Kosten.

2.7 Das Testumfeld berücksichtigen

Der Test ist an Einsatzgebiet und Umgebung des zu testenden Systems anzupassen. Die Testabdeckung, die Testendekriterien und der Testumfang sind bei jedem System neu zu bewerten und neu zu definieren. Die Konfiguration des Testsystems setzt entscheidende Rahmenbedingungen für die Durchführung der Testaktivitäten. Die spezifische Systemarchitektur, der Einsatzzweck und Anzahl der Anwender beeinflussen die Testaktivitäten. Sicherheitskritische Systeme oder Anwendungen in der Medizintechnik verlangen detailliertere und umfassendere Prüfungen als eine graphische Oberfläche einer App oder einer Spielesoftware. Auch die zu tolerierenden Fehler richten sich nach Art der getesteten Applikation.

2.8 Unterschiedliche Personen als Tester und Programmierer

Das Testen von Software ist eine eher destruktive Aufgabe. Man muss an das Produkt anders herangehen als der Entwickler. Dazu kommt, dass man das eigene Produkt naturgemäß milder betrachtet als ein neutraler Beobachter. Ein Programmierer, der sein eigenes Programm testet, findet prinzipiell weniger Fehler, da er versucht zu beweisen, dass sein Programm keine Fehler enthält. Fehler, die durch falsch verstandene Anforderungen entstehen, können nur durch außen stehende Personen aufgedeckt werden. Gerade weil zum Testen auch teilweise andere theoretische Grundlagen beherrscht werden müssen als für die Entwicklung von Software, sollten Tester und Programmierer aus unterschiedlichen Mitarbeitern bestehen.

2.9 Dokumentation und Nachvollziehbarkeit der Tests

Eingangsdaten, erwartete Ergebnisse und die erhaltenen Daten müssen dokumentiert werden. Das dient der Reproduzierbarkeit der Testaktivitäten: Bei gleichen Eingangsdaten muss der Test auf die gleichen Ergebnisse kommen.

Wenn dem nicht so ist, deutet das darauf hin, dass es noch nicht erhobene Parameter und Nebenbedingungen gibt, die für den Softwaretest ebenfalls zu berücksichtigen sind.

2.10 Operation gelungen, Patient tot?

Selbst wenn man strikt nach den Systemanforderungen testet und keine Fehler findet, heißt es noch lange nicht, dass das System den Nutzererwartungen entspricht. Wenn die Anforderungen an der Realität vorbeigehen oder zu umständlich sind, wird das System keine Nutzerakzeptanz finden, obwohl man doch eigentlich alles richtig gemacht hat. Daher empfiehlt es sich, den Endnutzer frühzeitig einzubinden, wo es möglich ist Prototypen zu entwickeln und das System im Pilotbetrieb zu validieren.

Gerade wenn man beim Test strikt nur von Requirements ausgeht, ohne nach links und rechts zu sehen und das Umfeld in der Praxis hinreichend berücksichtigt, läuft man Gefahr, dass man zwar formal der Software einem umfassenden Test unterzogen hat, aber dann am Ende ein Produkt ausliefert, das an den wahren Wünschen des Anwenders vorbeigeht.

2.11 Ermittlung der Fehlerrate

Erfahrungsgemäß treten Fehler oft gehäuft an bestimmten Abschnitten auf. Daher sollten Fehler vor allem in den Code-Segmenten gesucht werden, in denen schon viele Fehler aufgetreten sind. Beim Testen ist es hilfreich, einen Anhaltspunkt zu haben, wie viele Fehler eine bestimmte Software-Komponente wahrscheinlich beinhaltet. Um solche Schätzzahlen angeben zu können, gibt es mehrere Gleichungen als Ergebnis empirischer Untersuchungen.

Um diese Formeln zu bestimmen, wurde eine Vielzahl unterschiedlicher abgeschlossener Software-Projekte ausgewertet und die dabei erhobenen Daten mit einer mathematischen Relation angenähert. Werden diese Formeln auf ein neues Projekt angewendet, erhält man einen ungefähren Schätzwert zur Fehlererwartung.

Nach Basili kann man die Zahl der zu erwartenden Fehler wie folgt ermitteln:

$$\mathrm{F} = 4{,}04 + 0{,}0014\,\mathrm{LOC}^{4/3}.$$

Dabei ist LOC die Programmlänge in Codezeilen (*engl. lines of code*) und F die zu erwartende Fehlerzahl. Der geplottete Graph dazu ist in Abb. 2.1 zu sehen.

Um den Wert für LOC zu ermitteln, zählt man jede Zeile im Code des Testobjekts außer Leerzeilen und Kommentaren. Nach obiger Formel erhält man also für ein Programm mit 1000 „effektiven“ Programmzeilen eine erwartete Fehlerzahl von **17,7 Fehlern**.

Eine andere Formel zur Abschätzung der zu erwartenden Fehlerzahl lautet:

$$\mathrm{F_r} = \mathrm{C_1} + \mathrm{C_2}\ (\mathrm{SCHG/KLOC}) - \mathrm{C_3ISKL} - \mathrm{C_4}\ (\mathrm{DOCC/KLOC}).$$