Xpert.press

Klaus Franz

Handbuch zum Testen von Web- und Mobile-Apps

Testverfahren, Werkzeuge, Praxistipps

2. Auflage

Springer Vieweg

Xpert.press

Weitere Bände in dieser Reihe
http://www.springer.com/series/4393

Die Reihe Xpert.press vermittelt Professionals in den Bereichen Soft wareentwicklung, Internettechnologie und IT-Management aktuell und kompetent relevantes Fachwissen über Technologien und Produkte zur Entwicklung und Anwendung moderner Informationstechnologien.

Klaus Franz

Handbuch zum Testen von Web- und Mobile-Apps

Testverfahren, Werkzeuge, Praxistipps

2., aktualisierte und erweiterte Auflage

Springer Vieweg

Klaus Franz
Groß-Gerau
Deutschland

ISSN 1439-5428
ISBN 978-3-662-44027-8 ISBN 978-3-662-44028-5 (eBook)
DOI 10.1007/978-3-662-44028-5

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 Vieweg ist eine Marke von Springer DE. Springer DE ist Teil der Fachverlagsgruppe Springer Science+Business Media
www.springer-vieweg.de

Für meine Familie
Inge und Daniel

Vorwort zur zweiten Auflage

Nachdem die erste Auflage herausgegeben ist, werden fortlaufend neue Projekterfahrungen, neue Testmethoden und neue Technologien schriftstellerisch für die zweite Auflage des Handbuches zum Testen von Web-Applikationen verarbeitet. Diese ist also immer auf einem aktuellen Stand und kann gedruckt werden, sobald die erste Auflage vergriffen ist – just in time sozusagen.

Der Plan ist gut, klappt aber nicht, wenn zum einen das Autorendasein ein Hobby ist und zum anderen sich die IT-Technik rasant weiterentwickelt. Aber nun ist die zweite Auflage geschafft, erweitert um neue Inhalte.

Mobile Endgeräte haben den Markt und das Testen der darauf laufenden Software-Anwendungen in den letzten Jahren stark beeinflusst. Daher hat das Testen für und mit mobilen Geräten in vielen Kapiteln Einzug gehalten. Folgerichtig hat sich der Buchtitel geändert in „Handbuch zum Testen von Web- und Mobile-Apps“. Neben dem neuen Titel hat die zweite Auflage weitere Änderungen erfahren:

- Inhalt und Checklisten sind generell aktualisiert, insbesondere mit Blick auf die Anforderungen der mobilen Welt.
- Neue Tools werden genannt, einige sind aus dem Buch verschwunden.
- Es gibt ein neues Kapitel zur Testfallentwurfsmethode „Paarweises Testen“.
- Der Interoperabilitätstest ist mit Cross-Browser-Test und Cross-Device-Test zu einem Schwerpunktthema geworden.
- Neue Kapitel befassen sich mit Webservice-Test, Mobile-Funktionstest, Mobile-Installationstest, Mobile-Performanz-/Lasttest und Mobile-Zuverlässigkeitstest.
- Tests zu den Qualitätsmerkmalen Effizienz und Zuverlässigkeit sowie die Planung des Testteams sind in separaten Hauptkapiteln gelandet.
- Agile Software-Entwicklung und aktuelle Themen wie Cloud- und Crowd-Testing haben Einzug in die Testplanung (Teil III) gehalten.

Mein besonderer Dank geht an meinen Kollegen Dirk O. Schweier, der mir mit seinem fachkundigen Rat geholfen hat, dieses Buch fertig zu stellen, an meinen Sohn Daniel, der das Korrekturlesen während seiner Masterarbeit untergebracht hat und an meine Frau

Inge, die mein Autorendasein ertragen musste und auch noch die Rechtschreibung überprüft hat.

Über Verbesserungsvorschläge zum Buch und Tipps für neue Checklisten würde ich mich wieder sehr freuen.

Groß-Gerau Klaus Franz
Juli 2014 webtesting.klaus.franz@gmx.de

Vorwort zur ersten Auflage

Auslöser zu diesem Buch war meine Suche nach Informationen zum Testen von web-basierten Anwendungen. Seit vielen Jahren bin ich in der Qualitätssicherung in der Anwendungsentwicklung tätig, erst auf Großrechnern, dann auf Client-Server-Systemen. In diesen Umgebungen haben sich die „klassischen" Testverfahren seit langem bewährt. Aber was muss darüber hinaus beim Testen von Web-Anwendungen getan werden? Müssen die etablierten Methoden den neuen Technologien angepasst werden? Gibt es neue Verfahren und Hilfsmittel? Welche Testtools sind besonders im Web-Umfeld von Bedeutung?

Ich habe keine Literatur ausfindig machen können, die mir persönlich in kompakter Form zufriedenstellende Antworten auf diese Fragen geben konnte. Daher habe ich in diesem Buch Methoden, Hilfsmittel und Werkzeuge, die zur Qualitätssicherung von Web-Anwendungen beitragen, zusammengestellt.

Mit diesem Buch möchte ich ein Nachschlagewerk zum Testen von – nicht nur – web-basierten Anwendungssystemen bereitstellen, das zum einen auf die neuen Technologien eingeht und zum anderen die klassischen Testverfahren nicht aus den Augen verliert.

Das Schreiben eines Buches hat unverkennbare Parallelen zu einem Software-Entwicklungsprojekt: Der Autor (Projektleiter) hat Anforderungen an sein zu verfassendes Werk und gibt sich unter Berücksichtigung aller bekannten Risiken einen Zeitplan vor. Wie in (fast) jedem IT-Projekt verändern sich die Anforderungen und auch die Rahmenbedingungen. Und natürlich treten alle bekannten Risiken ein. Ressourcenengpässe und Terminverschiebungen sind die Folge.

Daher möchte ich mich bei meiner Familie und dem Springer-Verlag bedanken, die das Projekt in entscheidenden Momenten unterstützt und mir bei meinen Aktivitäten zur Seite gestanden haben.

Mein besonderer Dank geht an meine Kollegen und Freunde Siegfried Brauer, Klaus Gockel und Jochen Schneider, die mir mit ihrem fachkundigen Rat geholfen haben, dieses Buch fertig zu stellen, sowie an meinen Sohn Daniel, der seine Weihnachtsferien für das Korrekturlesen geopfert hat.

Über Verbesserungsvorschläge zum Buch und Tipps für neue Checklisten an webtesting.klaus.franz@gmx.de würde ich mich sehr freuen.

Groß-Gerau Klaus Franz
Januar 2007

Inhaltsverzeichnis

Teil III Testplanung

Abkürzungsverzeichnis

AbI	Aktionsbündnis für barrierefreie Informationstechnik
AGB	Allgemeinen Geschäftsbedingungen
AIR	Adobe Integrated Runtime
AJaX	Asynchronous JavaScript and XML
API	Application Programming Interface
ASP	Active Server Pages
BMAS	Bundesministerium für Arbeit und Soziales
BaNu	Barrieren finden, Nutzbarkeit sichern
BattV	Deutsche Batterieverordnung
BDSG	Bundesdatenschutzgesetz
BSI	Bundesamt für Sicherheit in der Informationstechnik
BITV	Barrierefreie-Informationstechnik-Verordnung
CASE	Computer Aided Software Engineering
CI	Continuous Integration
CIAC	Computer Incident Advisory Capability
CMMI	Capability Maturity Model Integration
COM	Common Object Model
CSS	Cascading Style Sheet
CTC	Click to Call
Daemon	Disk And Execution MONitor
DDoS	Distributed Denial of Service Angriff
DL-InfoV	Dienstleistungs-Informationspflichten-Verordnung (Verordnung über Informationspflichten für Dienstleistungserbringer)
DNS	Domain Name System
DoS	Denial of Service
EDGE	Enhanced Data Rates for GSM Evolution
EGG	Gesetz zum elektronischen Geschäftsverkehr
FAQ	Frequently Asked Questions
FMEA	Failure Mode and Effects Analysis
FTP	File Transfer Protocol
GjS	Gesetz über die Verbreitung jugendgefährdender Schriften und Medieninhalte

GPL	General Puplic License
GPRS	General Packet Radio Service
HSPA	High Speed Packet Access
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IuKT	Informations- und Kommunikationstechnologien
JAR	Java-Archiv
JDBC	Java Database Connectivity
JÖSchG	Gesetz zum Schutze der Jugend in der Öffentlichkeit
JSP	Java Server Pages
JMStV	Jugendmedienschutzstaatsvertrag
JuSchG	Jugendschutzgesetz
LDAP	Lightweight Directory Access Protocol
LFDN	Landesbeauftragte für den Datenschutz Niedersachsen
LTE	Long Term Evolution
MarkenG	Markengesetz
MitM	Man-in-the-Middle
MTBF	Mean Time Between Failure
MTTR	Mean Time To Repair
OWASP	Open Web Application Security Project
PAngV	Preisangabenverordnung
PDF	Portable Document Format
PHP	Private Home Page (Web-Programmiersprache)
QR	Queued Requests
QR-Code	Quick Response Code
QS	Qualitätssicherung
RIA	Rich Internet Applications
RPC	Remote Procedure Call
RPS	Requests Per Second
RPZ	Risikoprioritätszahl
RStV	Staatsvertrag über Rundfunk und Telemedien
RTMP	Real Time Messaging Protocol
SEO	Search Engine Optimization
SigG	Signaturgesetz
SigV	Signaturverordnung
SIM	Subscriber Identity Module
SOAP	Simple Object Access Protocol
SPICE	Software Process Improvement and Capability Determination
SQuaRE	Software product Quality Requirements and Evaluation
SSL	Secure Socket Layer
TKG	Telekommunikationsgesetz
TMG	Telemediengesetz

TPI	Test Process Improvement
UML	Unified Modeling Language
UMTS	Universal Mobile Telecommunications System
UrhG	Urheberrechtsgesetz
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
VM	Virtuelle Maschine
W3C	World Wide Web Consortium
WAI	Web Accessibility Initiative
WAR	Web-Archive
WAT	Web Accessibility Toolbar
WCAG	Web Content Accessibility Guidelines
Web	World Wide Web (Kurzform)
WSDL	Web Services Description Language
WWW	World Wide Web
XML	Extensible Markup Language
XHTML	Extensible Hypertext Markup Language
XSS	Cross Site Scripting
ZKDSG	Zugangskontrolldiensteschutz-Gesetz

Einleitung 1

Qualität ist das Maß der Nichtabweichung des Ists vom Soll.

Der Autor auf die Frage: „Was ist Qualität?“

1.1 Wieso dieses Buch?

Eine Software-Anwendung muss vielfältigen technischen und fachlichen Anforderungen genügen. Eine nicht vorhersagbare Anzahl von unbekannten Anwendern mit unterschiedlichem Fach- und Technikwissen greift weltweit mit diversen Browsern, Betriebssystemen und eventuell auch mit einer Vielzahl von mobilen Endgeräten auf eine Software zu. Die Qualitätssicherung einer Anwendung, sei sie nun mobil oder nicht, ist eine komplexe Aufgabe und muss professionell durchgeführt werden.

Der Einsatz der klassischen Testmethoden aus der Welt der Großrechneranwendungen, Transaktionsmonitore und Client/Server-Systeme ist auch für webbasierte und mobile Anwendungen notwendig, aber nicht hinreichend, um die geforderte Qualität sicherzustellen.

Die bekannten und erprobten Testmethoden müssen an die neuen Technologien angepasst und um neue Verfahren ergänzt werden. Genau dieses Ziel verfolgt das vorliegende Buch. Es soll eine umfassende, aber kompakte Arbeitshilfe zum Testen von Anwendungen – nicht nur – im Web- und im Mobile-Umfeld geben. Neben der Beschreibung der verschiedenen Testarten werden Hilfestellungen in Form von Tipps und Checklisten gegeben und die Einsatzmöglichkeiten von Testtools erläutert.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_1

1.2 Wem nutzt dieses Buch wie?

Dieses Buch richtet sich an Personen, die sich mit der Qualitätssicherung webbasierter und/oder mobiler Anwendungen befassen. Dabei kann es sich um informative Websites über kleine Mobile-Apps bis hin zu komplexen Webapplikationen zur Abwicklung von Geschäften über das Internet handeln[1]. Somit ist das Buch für Qualitätsmanager, Testmanager und Tester sowie qualitätsbewusste Entwickler, Projektleiter und Webmaster gleichermaßen interessant. Sie sollen mit dem Lesen dieses Buches folgende Lernziele erreichen:

- Sie können Qualitätsmerkmale, die eine Software-Anwendung erfüllen muss, beschreiben.
- Sie können Qualitätsanforderungen für ihre Web-/Mobile-Anwendung definieren.
- Sie können systematisch Testfälle für die Tests ihrer Software-Anwendung entwerfen.
- Sie können die zur Qualitätssicherung ihrer Anwendung notwendigen Testverfahren festlegen, priorisieren und anwenden.
- Sie wissen, welche Testtools bei welchen Tests eingesetzt werden können, und kennen deren Funktionsweise.
- Sie können im Rahmen ihrer Qualitätssicherungsmaßnahmen Risikoanalysen durchführen und Checklisten effektiv einsetzen.
- Sie wissen, wo weiterführende und nützliche Informationsquellen zum Testen zu finden sind.
- Sie wissen, welche Faktoren bei der Planung von Tests berücksichtigt werden müssen.

1.3 Wie ist dieses Buch zu lesen?

Dieses Buch konzentriert sich auf die Qualitätssicherungsmaßnahmen, die sich direkt auf die Qualität einer Software-Anwendung auswirken. Der Leser soll mit diesem Buch eine schnelle, effektive Unterstützung für seine durchzuführenden Tests erhalten.

Im ersten Teil des Buches werden die Grundlagen der Qualitätssicherung vermittelt. Im zweiten Teil, der sich insbesondere an die Praktiker richtet, werden Vorgehensweisen und Werkzeuge zum Testen beschrieben. Im dritten Teil werden Maßnahmen zur Planung und Steuerung der Tests aus Sicht des Testmanagers betrachtet.

1.3.1 Zum Teil I: Handwerkszeug

Wer die Begriffe aus der Welt der IT-Qualitätssicherung noch nicht beherrscht, der darf den ersten Teil dieses Buches nicht vernachlässigen, denn darin wird das Handwerkszeug

[1] Die Definitionen der Testobjekte sind Inhalt des Kap. 2.1.

eines Testers bereitgestellt. Dazu gehören die Normen zur Qualitätssicherung, die Definitionen zum Testen, die Testfallentwurfsmethoden, das Paarweise Testen, die Risikoanalyse und der Einsatz von Checklisten.

1.3.2 Zum Teil II: Testarten

Im zweiten Teil des Buches werden die einzelnen Testarten, die zur Qualitätssicherung eingesetzt werden, detailliert beschrieben. Zu jeder Testart werden wertvolle Tipps zum Vorgehen und zum Einsatz von Testwerkzeugen gegeben sowie Checklisten für die Praxis bereitgestellt. Mit diesen Informationen wird der Leser in die Lage versetzt, seine Tests zielgerichtet vorbereiten und effektiv durchführen zu können.

Wer das Handwerkszeug der Qualitätssicherung parat hat und sofort mit den Testarten starten möchte, um seinen Webauftritt oder seine Applikation zu testen, kann den ersten Teil des Buches überspringen und mit dem zweiten beginnen.

Die Tab. 1.1 zeigt eine Übersicht der im Teil II beschriebenen Testarten. In der Spalte „Checkliste“ sind die Checklisten aufgeführt, die zur jeweiligen Testart beschrieben sind.

Die folgenden Checklisten sind in Tab. 1.1 nicht aufgeführt, weil sie nicht direkt einer Testart zugewiesen werden können bzw. Planungsprozesse unterstützen:

Kapitel	Überschrift	Checkliste
4.3	Erfahrungsbasierte Testverfahren	Fehlererwartung
12.8	Planung der Effizienztests	Planung Effizienztest
16.3.1	Pilotbetrieb	Planung Pilotbetrieb

1.3.3 Zum Teil III: Testmanagement

Im dritten Teil des Buches werden die im Teil II beschriebenen Testarten aus Sicht des Testmanagers betrachtet, der die anstehenden Qualitätssicherungsmaßnahmen planen und steuern muss. Hier wird beschrieben, wie welche Testarten, Testmittel, Testressourcen und zeitlichen Testphasen im Rahmen der Testplanung auszuwählen, zu bewerten und einzusetzen sind.

Personen, die einen Überblick zur Planung der Tests erhalten möchten, können sich diesen im Teil III verschaffen.

1.3.4 Zum Glossar und Quellenverzeichnis

Fachbegriffe, die nicht im Text des Buches erklärt werden, aber eventuell nicht jedem Leser geläufig sind, sind im Text mit [GL] gekennzeichnet und im Glossar aufgenommen.

Tab. 1.1 Übersicht Testarten mit Checklisten

Kapitel	Testart	Checkliste
8	*Prüfungen von Dokumenten*	
8.1	Dokumententest	Dokument
		Objektmodell
9	*Tests zur Funktionalität*	
9.1	Unit-Test	
9.2	Funktionaler Komponententest	
9.3	Integrationstest	
9.4	Funktionaler Systemtest	
9.5	Link-Test	Link-Test
9.6	Cookie-Test	Cookie-Test
9.7	Plugin-Test	Plugin-Test
9.8	Test der Browser-Einstellungen	Browser-Einstellungen
9.9	Webservice-Test	
9.10	Sicherheitstest	Sicherheitstest
		Firewall (extern)
9.11	Interoperabilitätstest	
9.11.1	Cross-Browser-Test	
9.11.2	Cross-Device-Test	
9.12	Mobile-Funktionstest	Erkennung mobiler Geräte
10	*Tests zur Benutzbarkeit*	
10.1	Content-Test	Web-Content
		Webrecht
		Aufklärungspflicht
		Rechtliche Angaben
10.2	Oberflächentest	Oberflächentest
		Mobile-Oberflächentest
		Stichprobe Browser-Test,
		Stichprobe Mobile-Browser-Test
10.3	Usability-Test	
10.4	Zugänglichkeitstest	Zugänglichkeit WCAG 2.0 (extern)
10.5	Auffindbarkeitstest	Webadresse
		Barrierefreiheit und SEO-Freundlichkeit
11	*Tests zur Änderbarkeit und Übertragbarkeit*	
11.1	Code-Analysen	
11.1.1	Code-Walkthrough	
11.1.2	Code-Inspektion	Code-Inspektion

Tab. 1.1 (Fortsetzung)

Kapitel	Testart	Checkliste
11.1.3	Schreibtischtest	
11.1.4	Statische Code-Analyse durch Werkzeuge	
11.2	Installationstest	Installationstest
11.2.2	Mobile-Installationstest	Mobile-Installationstest
12	*Tests zur Effizienz*	
12.1, 12.6	Performanztest	Performanz Webseite
12.2	Last- und Stresstest	
12.3	Skalierbarkeitstest	
12.4	Speicherlecktest	
12.7	Mobile-Performanz-/Lasttest	Mobile-Performanztest
13	*Tests zur Zuverlässigkeit*	
13.1	Ausfallsicherheitstest	Ausfallsicherheitstest
13.2	Verfügbarkeitstest	
13.3	Mobile-Zuverlässigkeitstest	Mobile-Störfälle

Quellenangaben, die sich im Quellenverzeichnis wiederfinden, sind im Text in rechteckige Klammern […] gefasst. Dabei handelt es sich um Literaturquellen, Normen und Standards sowie Internet-Links. Letztere besitzen das Präfix URL ([URL: …]).

1.4 Übersicht Mobile-App-Testing

Ein neues Thema dieser zweiten Buchauflage sind die Tests im Umfeld mobiler Endgeräte. Viele mobile Applikationen[2] werden speziell für Smartphones und Tablet-Computer entwickelt und die „normalen" Websites müssen heutzutage mobiltauglich sein. Die Anzahl der zu testenden Hard- und Software-Komponenten wird immer größer. Spezielle Tools unterstützen die Tests für und mit mobilen Endgeräten.

Daher wird bei den beschriebenen Testverfahren, Checklisten, Testarten und Testtools auf die Besonderheiten des mobilen Testens[3] eingegangen und in vielen Fällen wird ihnen sogar ein eigenes Unterkapitel spendiert. Das bedeutet, dass es kein eigenes Hauptkapitel zum Mobile-App-Testing gibt, sondern dass sich das Thema an vielen Stellen im Buch wiederfindet. Tabelle 1.2 zeigt eine Übersicht, in welchen Kapiteln Ausführungen zum Mobile-App-Testing zu finden sind.

Oft wird der Begriff Mobile-Testing verwendet, wenn eigentlich Mobile-App-Testing gemeint ist; hier die Klarstellung:

[2] Definition Mobile-App s. Kap. 2.1

[3] …womit die Mobilität der Endgeräte gemeint ist und nicht die der Tester ☺.

Tab. 1.2 Übersicht Mobile-App-Testing

Kapitel	Titel	Mobile-Inhalt
2.1	Was soll qualitätsgesichert werden?	Definitionen Mobile-Webseite, Mobile-App und Mobile-Web-App
2.2.2	Spezielle Qualitätsmerkmale	Mobiltauglichkeit und Mobilfähigkeit
5.2.2	Paarweises Testen aus technischer Sicht	Paarweises Testen beim Mobile-App-Testing
9.11.2	Cross-Device-Test	Interoperabilitätstest mit mobilen Endgeräten
9.11.4	Werkzeuge für den Interoperabilitätstest	Tools zum Interoperabilitätstest
9.12	Mobile-Funktionstest	– Funktionstest von Desktop-Webseiten, Mobile-Webseiten, Native- und Hybrid-Apps, Mobile-Web-Apps – Mobiltauglichkeit – Erkennung mobiler Geräte – Spezielle Mobile-Funktionen – Werkzeuge für den Mobile-Funktionstest (W3C mobileOK Checker, Emulatoren, Simulatoren) ✓ Checkliste „Erkennung mobiler Geräte“
10.1.1	Erfüllung der Benutzererwartungen	✓ Checkliste „Mobile-Oberflächentest“ mit der Rubrik „Content auf mobilen Geräten“
10.2	Oberflächentest	Oberflächentest mit
		✓ Checkliste „Mobile-Oberflächentest“ und
		✓ Checkliste „Stichprobe Mobile-Browser-Test”
11.2.2	Mobile-Installationstest	Betrachtung der mobilen Aspekte beim Installationstest
		✓ Checkliste „Mobile-Installationstest“
12.7	Mobile-Performanz-/Lasttest	Betrachtung der mobilen Aspekte beim Performanz-/Lasttest
		Rail Driven Testing
		✓ Checkliste „Mobile-Performanztest“
12.8	Planung der Performanz-/Lasttests	✓ Checkliste „Planung Effizienztest – Mobiles Umfeld“
12.9	Werkzeuge für Effizienztests	Mobile-Performanz-/Lasttest mit NeoLoad

Tab. 1.2 (Fortsetzung)

Kapitel	Titel	Mobile-Inhalt
13.3	Mobile-Zuverlässigkeitstest	Behandlung von Störungen durch andere Anwendungen, Benutzer oder technische Gegebenheiten
		✓ Checkliste „Mobile-Störfälle“
15.2.5	Bereitstellung der mobilen Endgeräte	Wo kommen die mobilen Endgeräte her (intern, extern, Cloud)?
17.2	Die Test-Crowd	Crowdtests für mobile Anwendungen

- **Mobile-Testing** ist der Test der mobilen Endgeräte, d.h. der physikalischen Geräte als solche.
- **Mobile-App-Testing** ist der Test der Anwendungssoftware auf den mobilen Geräten.

1.5 Welche Testwerkzeuge werden genannt?

Für fast jede in diesem Buch beschriebene Testart werden Testwerkzeuge genannt. Dabei handelt es sich um eine subjektive Auswahl von Werkzeugen, die ich aus meiner Projektarbeit kenne. Das kann bedeuten, dass Werkzeuge, die vielleicht für die jeweils beschriebene Aufgabe genauso gut geeignet sind, nicht namentlich aufgeführt werden.

Umfassende und aktuelle Übersichten von frei verfügbaren und kommerziellen Testwerkzeugen sind zu finden unter:

- [URL: TestToolsJava]
- [URL: TestToolsOPENSOURCE]
- [URL: TestToolReview]
- [URL: TestToolsSQA]

1.6 Was liefert dieses Buch nicht?

Aussagen wie „*Spät gefundene Fehler sind die teuersten*“ sind richtig, bekannt, schon häufig diskutiert und ausführlich beschrieben. Daher werde ich auf Beweisführungen, warum und wieso Software rechtzeitig und intensiv getestet werden muss, in diesem Buch verzichten.

Auf Vorgehensmodelle (wie V-Modell XT[GL] und SCRUM[GL]), Prozessverbesserungsmodelle (wie CMMI®, SPICE, TPI NEXT®) und spezielle Themen zum Testmanagement (wie Konfigurations- und Fehlermanagement) gehe ich nur so weit ein, wie es zum Verständnis der beschriebenen Inhalte notwendig ist. Dass diese Themen etwas stiefmütterlich behandelt werden, liegt nicht daran, dass ich sie für unwichtig halte. Im Gegenteil,

jeder, dem Software-Qualität am Herzen liegt, muss sich diese Themen auf die Fahne schreiben. Denn Qualität von Software kann nur erzeugt werden, wenn alle Software-Produktionsprozesse geplant und gesteuert werden, ineinander greifen und einer permanenten Verbesserung unterliegen.

Weil diese Themen den Rahmen dieses Buches sprengen würden, verweise ich an dieser Stelle auf weiterführende Literatur:

- V-Modell XT → [URL: V-Modell]
- SRUM → [URL: SCRUM]
- TPI NEXT → [SOGETI_2011]
- CMMI → [Kneuper_2007]
- SPICE → [Hörmann_2006]
- Testmanagement → [Spillner_2011]

1.7 Wer sollte das Buch unbedingt lesen? Oder: Motivation für autofahrende Qualitätsskeptiker

Bei der Entwicklung und Einführung neuer Kraftfahrzeuge sind intensive Qualitätssicherungsmaßnahmen selbstverständlich, in der Software-Entwicklung leider nicht immer.

Nicht selten sind Entwickler und IT-Manager anzutreffen, die das Testen von Software für Zeit- und Geldverschwendung halten. Diesen Personen ist der folgende Vergleich gewidmet, in der Hoffnung, dass sie mehr Sensibilität gegenüber der Qualitätssicherung im IT-Projektgeschäft entwickeln. Wie würden sie reagieren, wenn ihr neues Auto ungeprüft ausgeliefert würde, sie die meisten Mängel erst selbst beim Fahren feststellen würden und aufgrund dessen Dauerkunden in ihrer KFZ-Werkstatt wären?

Eine Qualitätssicherungsanalogie

Ein neues PKW-Modell mit dem Arbeitstitel „WaF-Brauser“ (WaF = Work and Family) soll den Automobilmarkt bereichern. Das Management entscheidet, dass ein sportlicher Kleintransporter mit Schiebedach eine Marktlücke schließen soll.

Der neue, etwas luxuriöse WaF-Brauser soll für kleine Firmen interessant sein, deren Inhaber ihr Auto sowohl geschäftlich als auch privat nutzen.

Die Anforderungen an den WaF-Brauser werden zusammengetragen. Dazu gehören:

- Mindestens fünf Sitze
- Große Ladefläche
- Flott beschleunigender Dieselmotor mit geringem Verbrauch
- Hoher Sicherheitsstandard
- Optional: hochwertiges, flexibel einsetzbares Car Multimedia System mit Internetverbindung zur Navigationsunterstützung

→Review
Mit diesen Angaben werden die Konstrukteure konfrontiert. Diese sind nicht glücklich damit und stellen viele Fragen:

- Was heißt „mindestens fünf Sitze“? Können es auch sechs, sieben oder neun Sitze sein?
- Müssen die „große Ladefläche“ und die Anzahl von fünf Sitzen gleichzeitig möglich sein oder kann die „große Ladefläche“ auf Kosten ausgebauter Sitze gehen?
- Was genau ist eigentlich eine „große Ladefläche“?
- Und was ist „flotte Beschleunigung“ mit „geringem Verbrauch“?
- Hoher Sicherheitsstandard bedeutet was?
- Was gehört zu einem hochwertigen, flexiblen Car Multimedia System? Navi? Freisprechanlage? Fahrerassistenzsysteme? Rückfahrkamera? USB-Anschluss? Soundsystem?
- Welche externen Geräte sollen am Navi anschließbar sein? Smartphones, mobile Navis, USB-Sticks, Speicherkarten, Kameras?

→Qualitätsmerkmale
Die Fragen zeigen die Problematik für die Konstrukteure. Es sind zwar Merkmale genannt, die der neue WaF-Brauser erfüllen soll, aber keine konkreten Vorgaben. Um konkrete Konstruktionspläne erstellen zu können, werden detailliertere Informationen benötigt.

→Qualitätsanforderungen
Die Auftraggeber gehen also in Klausur und machen sich Gedanken über die Details. Konstrukteure und Designer werden in die Arbeitsgruppe mit aufgenommen. Als Ergebnis sind die Anforderungen nun mit eindeutig messbaren Größen beschrieben:

- Der Laderaum muss eine Höhe von mindestens 1,85 m haben.
- Bei einer freien Ladefläche von mindestens 2,20 × 1,50 m müssen 6 Personen transportiert werden können.
- Bei einer um 80 % vergrößerbaren Ladefläche müssen noch 3 Personen transportiert werden können.
- Der durchschnittliche Kraftstoffverbrauch ohne Beladung soll bei maximal 6,5 L Dieselkraftstoff pro 100 km liegen.
- Die standardisierten Crash-Tests müssen den Nachweis der höchsten Sicherheitsstufe (fünf Sterne) erbringen.
- Das Car Multimedia System soll in der Standardversion enthalten: Navi, Soundsystem, Freisprechanlage via Bluetooth-Kommunikation (für eine Anzahl konkret definierter Mobilgeräte). Optional bestellbar sind: (in einem separaten Dokument definierte) Fahrerassistenzsysteme, Rückfahrkamera, Internet-Fähigkeit, USB-3 Anschluss, DVD-Player.

→Dokumententest
In der nächsten Entwicklungsphase werden die Baupläne konstruiert, das Design wird entworfen. Natürlich werden diese Dokumente und Pläne mehreren Prüfungen durch Experten unterzogen.

→Usability-Test
Bevor der WaF-Brauser in Produktion geht, wird ein Prototyp entwickelt und potentiellen Kunden vorgeführt. Diese sollen zum Beispiel folgende Fragen beantworten:

- Sind die Sitze bequem?
- Ist die Ladefläche leicht zu beladen?
- Ist das Cockpit für kleine Personen überschaubar?
- Erlaubt die Öffnung der Heckklappe das Beladen mit großen Gegenständen?
- Sind alle Knöpfe und Schalter ergonomisch angeordnet?
- Ist das Car Multimedia System intuitiv bedienbar?

Die Testpersonen dürfen Probe fahren und werden anschließend über ihre Meinungen und Erwartungen befragt. Es ergeben sich neue Anforderungen:

- Bis zu 8 Sitzplätze sollen verfügbar sein.
- Dachreling und Navigationssystem sollen Standard sein.
- Standheizung soll nur auf Wunsch ausgeliefert werden und nachrüstbar sein.
- Einen Kühlschrank braucht niemand.
- Große Schiebetüren werden seitlich hinten gefordert.
- Die Rückfahrkamera muss aus Sicherheitsgründen in der Basisausstattung sein.
- Die Navi-Bedienung versteht niemand so recht, auch nicht mit Handbuch.

→Komponententest
Nachdem alle Anforderungen erneut beschrieben, geprüft und besiegelt sind, kann der WaF-Brauser produziert werden.

Ein Auto besteht aus einer großen Anzahl von Bausätzen (Komponenten). Von jedem internen und externen Lieferanten wird gefordert, dass er seine Komponenten nachweisbar qualitätsgesichert hat. Sowohl für jedes Einzelteil eines Bausatzes (z. B. Bremsschlauch) als auch für das Zusammenspiel der Einzelteile (die Bremse als Ganzes) muss eine Kontrolle nachgewiesen werden.

→Integrationstest
Das gilt auch für die Verträglichkeit (Bremsflüssigkeit) und Kommunikationsfähigkeit mit anderen Komponenten (z. B. ABS).

→Fehlernachtest und Regressionstest
Bei den Prüfungen stellt sich heraus, dass Mikroprozessoren eines bestimmten Typs fehlerhaft arbeiten. Nach Lieferung neuer Prozessoren werden alle Komponenten, in die sie integriert sind, neuen Prüfungen unterzogen.

→Systemtest
Die Komponenten werden nun sukzessive zusammengesetzt und nach jeder Phase überprüft, bis am Ende ein WaF-Brauser fertiggestellt ist. Einige Exemplare werden von Werksfahrern auf kurzen Strecken und unter normalen Bedingungen Probe gefahren. Dabei werden alle möglichen Funktionen ausprobiert.

→Interoperabilitätstest
Die Freisprechanlage via Bluetooth funktioniert einwandfrei mit dem Blackberry® 9300 des Testers. Aber wie verhält es sich mit den anderen Mobiltelefonen mit ihren verschiedenen Betriebssystemen und Herstellern? Es gibt gefühlt 124 wichtige, unterschiedliche Endgeräte. Für welche kann die Funktionalität der Freisprechanlage gewährleistet werden? Die einsetzbaren Geräte werden festgelegt, getestet und dokumentiert.

→Pilotphase
Anschließend dürfen einige gute Kunden den WaF-Brauser im täglichen Gebrauch fahren und bewerten. Aufgrund der bisher hervorragenden Qualitätssicherungsmaßnahmen treten nur kleine Mängel zu Tage, die schnell im Produktionsprozess eliminiert werden können.

→Effizienztest
Bevor der WaF-Brauser in großer Stückzahl in Serie gehen kann, sind noch einige Tests durchzuführen. Das Fahrzeug muss auf längeren Strecken beweisen, dass Beschleunigung, Kraftstoffverbrauch, Fahrgeräusche und Sitzkomfort den Qualitätsanforderungen entsprechen.

→Lasttest
Weiterhin muss sichergestellt sein, dass sich das Fahrzeug in extremen Situationen erwartungsgemäß verhält:

- Der Wagen läuft auch bei Höchstgeschwindigkeiten ruhig.
- Die Beschleunigung ist in kritischen Situationen hinreichend.
- Der Öl- und Kraftstoffverbrauch überschreitet auf längeren Schnellfahrten nicht die angegebenen Werte.
- Straßenlage und Bremsverhalten entsprechen bei hohem Tempo und bei voller Beladung den Vorgaben (Elchtest).

→Speicherlecktest
Es gehen keine Betriebsstoffe (Sprit, Öl, Wasser, Bremsflüssigkeit) verloren.

→Ausfallsicherheitstest
Wenn wichtige Komponenten kurz vor dem Versagen stehen oder ausgefallen sind, wird der Fahrer direkt informiert (ABS, ESP, Öldruck, …).

→Sicherheitstest
Die Sicherheit der Insassen muss gewährleistet sein:
- Bei einer Vollbremsung ohne Aufprall wird der Airbag nicht ausgelöst, sondern erst beim Auftreffen auf ein Hindernis ab der vorgegebenen Kraft.
- Der Crash-Test eines unabhängigen Gutachterinstituts erbringt die Sicherheitsstufe vier. Die Konstruktion der Fahrerkabine wird nachgebessert. Erst danach werden die fünf Sterne für höchste Sicherheit vergeben.
- Die Wegfahrsperre und die Diebstahlsicherung entsprechen den Normen.

→Zugänglichkeitstest
Wenn der WaF-Brauser für Rollstuhlfahrer geeignet sein soll, müssen dazu die gesetzlichen Vorschriften eingehalten und überprüft werden.

→Plugin-Test
Wie steht es mit den Sonderausstattungen? Dachgepäckträger, Heckträgersysteme für Fahrräder, Anhängerkupplungen und ähnliche Ergänzungen dürfen Fahrkomfort und Sicherheit nicht beeinträchtigen.

→Mobile-Zuverlässigkeitstest
Funktioniert die Mobile-App des Car Multimedia Systems zur Standortabfrage mit allen Netzwerkanbietern und dann auch bei hohen Geschwindigkeiten?

→Abnahmetest
Wird der Wagen erstmals für den Straßenverkehr zugelassen oder werden nachträglich zusätzliche Teile am Wagen eingebaut, ist eine TÜV-Abnahme erforderlich.

→Crowdtest
Als Marketingmaßnahme wird eine Lotterie gestartet, bei der die Gewinner ein Wochenende den WaF-Brauser kostenlos bei vollem Tank testen können.

→Regressionstest
Wenn der WaF-Brauser in Betrieb ist und gefahren wird, muss er regelmäßig zum TÜV. Dort wird wiederholt geprüft, ob der Wagen noch verkehrstauglich ist.

→Verfügbarkeitstest
In der Zentrale des Herstellers des WaF-Brausers werden Statistiken erhoben, wie oft ein Fahrzeugtyp mit Mängeln in der Werkstatt ist und wie lang die Reparaturzeiten sind, um

eventuell Verbesserungen für die nächste Version einzuplanen oder um notfalls Rückrufaktionen zu starten.

- Wenn Autos so nachlässig getestet würden, wie manche (oder viele?) Software-Anwendungen, würde die Menschheit zu Fuß gehen.

Teil I
Handwerkszeug

2. Definitionen zur Qualität
3. Begriffe zum Testen
4. Testfallentwurfsverfahren
5. Paarweises Testen
6. Risikoanalyse
7. Checklisten

Im ersten Teil des Buches werden die Grundlagen zur Qualitätssicherung von Software vermittelt. Dazu gehören die Definitionen zur Qualität und zum Testen, die Verfahren zum Entwurf von Testfällen, das Paarweise Testen, die Risikoanalyse und der Einsatz von Checklisten. Die Grundlagen werden für das Verständnis der Teile II und III dieses Buches benötigt und sie geben dem Tester das methodische Werkzeug für seine Arbeit in die Hand.

Testwerkzeuge gehören zwar auch zum Handwerkszeug eines Testers, werden aber in Teil II direkt bei jeder Testart beschrieben, die sie jeweils unterstützen.

Definitionen zur Qualität

2

Qualität ist kein Zufall;
sie ist immer das Ergebnis angestrengten Denkens.

John Ruskin (1819–1900)

In diesem Kapitel werden Definitionen zur Qualität und Maßnahmen beschrieben, mit denen die Qualität von – nicht nur – webbasierten und mobilen Applikationen sichergestellt wird. Dazu werden vier wichtige Fragen der Qualitätssicherung beantwortet:

1. Was soll qualitätsgesichert werden?
2. Was ist Qualität?
3. Wie kann Qualität gemessen werden?
4. Wie kann Qualität erzeugt werden?

2.1 Was soll qualitätsgesichert werden?

Ein **Testobjekt** ist u. a. ein Programm, eine Komponente, ein Teilsystem oder das gesamte System, das einem Test unterzogen wird.

Die Differenzierung von Testobjekten spielt eine wichtige Rolle. Es ist ein Unterschied für die einzusetzenden Methoden, Tools und Ressourcen, ob eine Website, eine Mobile-App oder eine Web-App qualitätsgesichert werden muss. Daher ist es wichtig, die Typen der zu testenden Objekte voneinander abzugrenzen.

Es gibt Systemsoftware, systemnahe Software und Anwendungssoftware. Systemsoftware sorgt dafür, dass Rechner überhaupt funktionstüchtig sind, wie z. B. Betriebssysteme. Systemnahe Software wird benötigt, um Anwendungssoftware realisieren zu können,

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_2

dazu gehören z. B. Compiler und Testtools. In diesem Buch konzentrieren wir uns auf Anwendungssoftware.

► Alles ist App!

In der IT ist eine **Anwendung** oder Applikation oder kurz **App** ganz allgemein eine Software, die auf (irgend)einem Computer läuft und die einen Benutzer bei einer bestimmten Aufgabe unterstützt und ihm einen Nutzen bringt. Eine Anwendung alias App kann

- als Batch-Job (Stapelverarbeitung) ohne Benutzerinteraktion auf einem Rechner laufen,
- als Embedded Software ein technisches System regeln, steuern und überwachen oder
- dialogorientiert über eine Benutzeroberfläche mit dem Benutzer kommunizieren.

Web-Apps und Mobile-Apps, die – wie der Titel verspricht – in diesem Buch schwerpunktmäßig betrachtet werden, sind dialogorientierte Anwendungen.

Eine **Web-App** ist ein Computerprogramm, das auf einem Webserver ausgeführt wird, wobei eine Interaktion mit dem Benutzer ausschließlich über einen Webbrowser erfolgt. Hierzu sind der Computer des Benutzers (Client) und der Server über ein Netzwerk, wie das Internet oder ein Intranet, miteinander verbunden, so dass die räumliche Entfernung zwischen Client und Server unerheblich ist. Eine Web-App kann per Definition auch im Browser eines mobilen Gerätes laufen.

Bei einer **Client-Server-App** kommuniziert der Client ebenfalls mit einem Server in einem Netzwerk. Aber im Gegensatz zur Web-App ist eine Client-Server-Anwendung plattformabhängig und muss auf dem Client installiert werden.

Der Vollständigkeit halber: Eine **Desktop-App** läuft nur lokal auf einem Desktop-PC (oder Notebook) und muss dort installiert werden.

Eine mobile Applikation oder kurz **Mobile-App** ist eine Anwendungssoftware für mobile Endgeräte (wie Smartphones und Tablet-Computer) bzw. für mobile Betriebssysteme. Mobile-Apps werden unterschieden in Mobile-Web-Apps, Native-Apps und Hybrid-Apps.

Mobile-Web-Apps basieren auf Webtechnologien, kommunizieren mit einem Server und werden über den Webbrowser des Mobilgeräts abgerufen, sie müssen daher nicht installiert werden. Eine Mobile-Web-App kann prinzipiell im Browser eines Desktop-PCs laufen, sofern die Bedienung möglich ist und sie nicht für Desktops gesperrt ist.

Native-Apps werden für ganz spezielle Geräte für ein bestimmtes Betriebssystem entwickelt, sie sind hersteller- und plattformabhängig und müssen installiert werden.

Hybrid-Apps vereinen die Vorteile von Mobile-Web-Apps und Native-Apps, weil sie auf die Hard- und Software-Komponenten des mobilen Endgerätes zugreifen und gleichzeitig unterschiedliche Plattformen bedienen können. Zum Beispiel können Kalender- und Kamerafunktionen in eine Hybrid-App integriert werden. Hybrid-Apps müssen installiert werden.

Fassen wir etwas salopp zusammen:

- Eine App mit „Web“ im Namen läuft auf browserfähigen Geräten, ist plattformunabhängig und muss (abgesehen von eventuell benötigten Plugins) nicht installiert werden.

- Eine App ohne „Web“ im Namen ist plattformabhängig, läuft nicht im Browser, besitzt daher eine eigene Benutzeroberfläche und muss auf dem Client installiert werden.

Websites liefern dem Benutzer Informationen, dienen der Kontaktaufnahme und können Anwendungen wie z. B. Webshops im Netz bereitstellen. Sie sind quasi abgespeckte Web-Apps.

Eine **Website** (Webpräsenz oder Webauftritt) ist die Gesamtheit aller Seiten eines Webangebotes, d. h. alle Webseiten bilden zusammen eine Website. Eine **Webseite** (Webpage) ist eine einzelne Internetseite, d. h. ein Teil einer Website. Eine **Homepage** ist die Startseite einer Website.

Die drei Testobjekte Website, Webseite und Homepage werden manchmal fälschlicherweise synonym verwendet. Als Merksatz zur Unterscheidung der drei Begriffe dient die Qualitätsanforderung[1]:

- Jede Webseite einer Website muss einen Link auf die Homepage enthalten.

Bezogen auf die Zielplattform können Webseiten weiter differenziert werden. Ist eine Webseite für Desktop-PCs und Notebooks programmiert, handelt es sich um eine **Desktop-Webseite**.

Eine mobile Webseite oder **Mobile-Webseite** ist speziell für die Darstellung von Inhalten auf mobilen Endgeräten entwickelt. Gewöhnlich stellt eine mobile Webseite als Ergänzung des Webauftritts den Inhalt einer Desktop-Webseite in grafisch abgespeckter, für mobile Geräte optimierter Form dar.

Geschafft! Wir sind bei der Antwort auf die Frage, welche Typen von Testobjekten in diesem Buch schwerpunktmäßig betrachtet werden. Es sind die Testobjekttypen, die der Benutzer „vordergründig“ auf seiner Anwendungsoberfläche, seinem **Front-End** sieht:

- Website
- Web-App
- Mobile-Web-App

Im Normalfall bestehen diese Testobjekte nicht alleine, sie sind Bestandteil eines mehr oder weniger umfangreichen Gesamtsystems. Denn im Hintergrund kommunizieren sie mit einer Vielzahl von Systemkomponenten, dem **Back-End**. Dazu können unter anderem

[1] Qualitätsanfoderungen s. Kap. 2.3.

Datenbanken, Firewalls, Endgeräte, Netzwerke, Server, Schnittstellen und Webservices gehören. Alle diese Objekte müssen „hintergründig" in die Tests miteinbezogen werden.

2.2 Was ist Qualität?

Was Qualität von Software ist und welche Merkmale die Qualität von Software bestimmen, wird in mehreren Normen festgelegt. Die für den Inhalt dieses Buches relevanten Normbegriffe werden im Folgenden vorgestellt.[2] Zudem werden spezielle Qualitätsmerkmale für die im Kap. 2.1 definierten Testobjekte des Front-Ends beschrieben.

2.2.1 Normenbestimmte Qualitätsmerkmale

Die ISO/IEC 25000 (Software-Engineering – Software product Quality Requirements and Evaluation (SQuaRE) [ISO/IEC 25000]) ist eine internationale, mehrteilige Norm. Sie ersetzt die bis 2005 gültige ISO/IEC 9126 (Software-Engineering – Qualität von Software-Produkten) und legt nun die Begriffe zur Software-Qualität fest:

> ► „Software-Qualität ist die Gesamtheit der Merkmale und Merkmalswerte eines Software-Produkts, die sich auf dessen Eignung beziehen, festgelegte oder vorausgesetzte Erfordernisse zu erfüllen."

Die Normen unterscheiden die sechs **Qualitätsmerkmale**

- Funktionalität,
- Zuverlässigkeit,
- Benutzbarkeit,
- Effizienz,
- Änderbarkeit,
- Übertragbarkeit,

wobei die letzten fünf die **nicht-funktionalen Qualitätsmerkmale** einer Software sind.

Nach diesen sechs Qualitätsmerkmalen sind die Kapitel des zweiten Teiles dieses Buches gegliedert, in denen die Tests zur Überprüfung eben dieser Merkmale beschrieben werden.

Funktionalität (Functionality) Vorhandensein von Funktionen, die festgelegte und vorausgesetzte Erfordernisse erfüllen.

[2] Die vollständigen Normtexte sind beim Beuth-Verlag [URL: Beuth] erhältlich.

Teilmerkmale: Angemessenheit, Interoperabilität[3], Ordnungsmäßigkeit[GL], Richtigkeit, Sicherheit[4]

Zuverlässigkeit (Reliability) Fähigkeit einer Software/eines Systems, ihr/sein Leistungsniveau unter festgelegten Bedingungen in einem festgelegten Zeitraum zu halten.

Teilmerkmale: Fehlertoleranz, Reife, Wiederherstellbarkeit

Benutzbarkeit (Usability) Fähigkeit einer Software, unter festgelegten Bedingungen für einen Benutzer verständlich, erlernbar, anwendbar und attraktiv zu sein.

Teilmerkmale: Attraktivität, Bedienbarkeit, Erlernbarkeit, Verständlichkeit

Effizienz (Efficiency) Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen.

Teilmerkmale: Verbrauchsverhalten, Zeitverhalten (Performanz)

Änderbarkeit[5] (Maintainability) Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist. Änderungen können Korrekturen von Fehlern, Anpassungen an veränderte Anforderungen oder eine veränderte Systemumgebung und die Verbesserung der Wartung einschließen.

Teilmerkmale: Analysierbarkeit, Modifizierbarkeit, Prüfbarkeit, Stabilität

Übertragbarkeit[6] (Portability) Leichtigkeit, mit der eine Software von einer Umgebung in eine andere übertragen werden kann. Umgebung kann die organisatorische Umgebung, die Hardware-Umgebung oder die Software-Umgebung bedeuten.

Teilmerkmale: Anpassbarkeit, Austauschbarkeit, Installierbarkeit, Koexistenz[GL]

Konformität ist ein Teilmerkmal, das zusätzlich alle sechs Qualitätsmerkmale besitzen. **Konformität** ist der Grad, in dem die Software Normen oder Vereinbarungen zum jeweiligen Qualitätsmerkmal erfüllt.

Neben der ISO/IEC 25000 definiert die DIN EN ISO 9241 („Ergonomie der Mensch-System-Interaktion“) Software-Qualitätsmerkmale. Für den Tester von Anwendungen mit einer Benutzerschnittstelle sind deren Teile 11 „Anforderungen an die Gebrauchstauglichkeit (Allgemeine Leitsätze)“, 110 „Grundsätze der Dialoggestaltung“, 151 „Leitlinien zur Gestaltung von Benutzungsschnittstellen für das World Wide Web“ und 171 „Richtlinien für die Zugänglichkeit von Software“ relevant.[7]

Wenn es darum geht, welche Inhalte auf einer Website erscheinen sollen, wie die Kundenzufriedenheit sichergestellt und die Anwendung optimal bedient werden kann, ist die

[3] Definition von Interoperabilität s. Kap. 9.11.

[4] Definition von Sicherheit s. Kap. 9.10.

[5] Änderbarkeit und Wartbarkeit werden synonym verwendet.

[6] Übertragbarkeit und Portabilität werden synonym verwendet.

[7] Siehe [ISO 9241-11], [ISO 9241-110], [ISO 9241-151], [ISO 9241-171].

ISO 9241 Teil 110 hilfreich. Sie belegt die „Grundsätze der Dialoggestaltung“ mit folgenden Merkmalen:

Aufgabenangemessenheit „Ein interaktives System ist aufgabenangemessen, wenn es den Benutzer unterstützt, seine Arbeitsaufgabe effektiv und effizient zu erledigen, d. h., wenn Funktionalität und Dialog auf den charakteristischen Eigenschaften der Arbeitsaufgabe basieren, anstatt auf der zur Aufgabenerledigung eingesetzten Technologie.“

Eine Webanwendung ist zum Beispiel aufgabenangemessen, wenn keine aufgabenfremden, unnötigen Eingaben gefordert werden, wie z. B. irrelevante, persönliche Angaben zu einer Bestellung.

Die Vermeidung unnötiger Eingaben dient der Aufgabenangemessenheit. Dazu gehört, dass Standardwerte automatisch für den Benutzer gesetzt werden, damit er sie nicht selber eintragen muss, z. B. Tagesdatum oder Ortsname zur eingegebenen Postleitzahl.

Erwartungskonformität „Ein Dialog ist erwartungskonform, wenn er den aus dem Nutzungskontext heraus vorhersehbaren Benutzerbelangen sowie allgemein anerkannten Konventionen entspricht.“

Eine Webanwendung ist erwartungskonform, wenn Funktionscodes und -tasten in allen Masken und Menüs gleich verwendet werden oder wenn unterstrichene Wörter stets *Hyperlinks* sind.

Fehlertoleranz „Ein Dialog ist fehlertolerant, wenn das beabsichtigte Arbeitsergebnis trotz erkennbar fehlerhafter Eingaben entweder mit keinem oder mit minimalem Korrekturaufwand seitens des Benutzers erreicht werden kann.“

Eine Webanwendung ist zum Beispiel fehlertolerant, wenn die Eingaben in ein Formular vor dem Abschicken überprüft werden und nur die fehlerhaften Eingaben erneut erfasst werden müssen.

Oder ein Eingabefeld erkennt eine fehlerhafte Eingabe automatisch und teilt dies dem Benutzer mit. Trotzdem kann der Benutzer seine Arbeit erst einmal fortsetzen.

Individualisierbarkeit „Ein Dialog ist individualisierbar, wenn Benutzer die Mensch-System-Interaktion und die Darstellung von Informationen ändern können, um diese an ihre individuellen Fähigkeiten und Bedürfnisse anzupassen.“

Eine Webanwendung ist zum Beispiel individualisierbar, wenn der Nutzer die Inhalte für seinen persönlichen Newsletter bestimmen oder sein persönliches Benutzerprofil speichern kann. Abschaltbare bzw. erweiterbare Symbolleisten oder Menüs gehören ebenfalls zur Individualisierung.

Lernförderlichkeit „Ein Dialog ist lernförderlich, wenn er den Benutzer beim Erlernen der Nutzung des interaktiven Systems unterstützt und anleitet.“

Durchgängige Konzepte bei der Strukturierung von Dialogen sind lernförderlich, z. B. existiert ein Menüpunkt, in dem alle Funktionen aufgelistet sind, und für jeden Funktionsaufruf wird stets der Anfangsbuchstabe mit der Strg-Taste als Shortcut verwendet.

Eine Webanwendung ist zum Beispiel lernförderlich, wenn eine Guided Tour durch die Anwendung existiert und Testbuchungen vorgenommen werden können.

Selbstbeschreibungsfähigkeit „Ein Dialog ist in dem Maße selbstbeschreibungsfähig, in dem für den Benutzer zu jeder Zeit offensichtlich ist, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können."

Eine gute selbstbeschreibende Website gibt dem Benutzer Antworten auf die großen Wo-Fragen:

- Wo komme ich her?
- Wo bin ich?
- Wo kann ich hin?
- Wo finde ich Hilfe?

Auf Websites mit vielen Seiten sind Breadcrumbs[GL] (Brotkrumen) ein wichtiger Bestandteil der Selbstbeschreibung. Eine Breadcrumb-Navigation zeigt dem Nutzer seinen aktuellen Standort in der Webanwendung. Die Brotkrumen reduzieren benutzerfreundlich die Anzahl der Aktionen eines Website-Besuchers, die ihn auf eine Seite einer höheren Ebene bringen – schönen Gruß von Hänsel und Gretel.

Beispiele für Selbstbeschreibung

Ein Beispiel einer selbstbeschreibenden Website findet man auf https://www.deutschland.de, wo neben dem aktuellen Standort (Breadcrumb) auf der Website auch die Anzahl der Beiträge auf der aktuellen Webseite angezeigt wird, s. Abb. 2.1.

Eine Webanwendung ist selbstbeschreibend, wenn die Anzahl der Treffer nach einer Suche wie in Abb. 2.2 angezeigt wird oder wenn das Eingabeformat eines Datumfeldes vorgegeben ist.

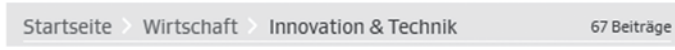


Abb. 2.1 Breadcrumbs

Abb. 2.2 Anzeige Suchtreffer

Ein reales Beispiel für „Nicht-Selbstbeschreibung“

Und so sollte man es nicht machen: Ich erfasse in einem Webformular meine Telefonnummer mit Leerzeichen nach der Vorwahl <0999 888888>, denn es sind keine Angaben zur Erfassung auf der Seite gemacht. Ich erlebe eine Überraschung:

```
------------------------------------------------
Fehler beim Ändern der Kundendaten

Hinweis: Bitte ergänzen bzw. berichtigen Sie
folgende Angaben:
Fehl-SC = „61“
Fehl-SC-Text = „Schreibweise fehlerhaft“
------------------------------------------------
```

Was habe ich falsch gemacht? Vielleicht das Leerzeichen rausnehmen: <0999888888>? Aha! Jetzt wird mir mitgeteilt, in welchem Format die Telefonnummer eingegeben werden muss:

```
------------------------------------------------
Datenänderung

Hinweis: Bitte ergänzen bzw. berichtigen Sie
folgende Angaben:
Tel. privat ist nicht korrekt (Bitte Format
Vorwahl/Rufnummer verwenden)
------------------------------------------------
```

Wie schön wäre es doch, stände im Label „Tel. privat (Vorwahl/Rufnummer)“ oder wären zwei separate Felder für Vorwahl und Rufnummer festgelegt.

Steuerbarkeit „Ein Dialog ist steuerbar, wenn der Benutzer in der Lage ist, den Dialogablauf zu starten sowie seine Richtung und Geschwindigkeit zu beeinflussen, bis das Ziel erreicht ist.“

Eine Webanwendung ist zum Beispiel steuerbar, wenn für Suchvorgänge die Anzahl der anzuzeigenden Treffer pro Seite festgelegt, Download-Prozesse unterbrochen und Funktionen mit Tastenkürzeln statt über Menüs aufgerufen werden können.

Ein Beispiel für Steuerbarkeit

Ein positives Beispiel für Steuerbarkeit zeigt Abb. 2.3: Steuerbarkeit. Dort kann der Benutzer unter <Optionen> – <Sucheinstellungen> unter anderem selbst festlegen, ob Sucheingaben vervollständigt werden, wie viele Treffer pro Seite und wo sie angezeigt werden sollen.

Abb. 2.3 Steuerbarkeit. (Quelle: www.google.de/preferences?hl=de)

Gebrauchstauglichkeit Die Gebrauchstauglichkeit (Usability) einer Anwendung wird in der ISO 9241 – Teil 11 [ISO 9241-11] definiert:

„Das Ausmaß, in dem ein Produkt durch bestimmte Benutzer in einem bestimmten Nutzungskontext genutzt werden kann, um bestimmte Ziele effektiv, effizient und zufriedenstellend zu erreichen."

Und weiter erklärt die Norm

- **Effektivität** als die Genauigkeit und Vollständigkeit, mit der Benutzer ein bestimmtes Ziel erreichen,
- **Effizienz** als den im Verhältnis zur Genauigkeit und Vollständigkeit eingesetzten Aufwand, mit dem Benutzer ein bestimmtes Ziel erreichen,
- **Zufriedenstellung** als die Freiheit von Beeinträchtigung und positive Einstellung gegenüber der Nutzung des Produkts.

Die aufgeführten Anforderungen an die Dialoggestaltung einer Software und die Gebrauchstauglichkeit sind dem Qualitätsmerkmal Benutzbarkeit zuzuordnen. Ihnen wird insbesondere im Usability-Test (s. Kap. 10.3) Rechnung getragen.

Ein Beispiel für Gebrauchsuntauglichkeit

Nicht gerade gebrauchstauglich ist der Onlineshop (m)eines Hardware-Händlers:

Ich interessiere mich für Monitore. In die engere Auswahl kommen drei Exemplare. Damit ich diese schnell wiederfinde, möchte ich sie mir in einer Wunschliste speichern. Ich weiß, dass es die Wunschliste gibt, ich kann mich noch vom letzen Einkauf daran erinnern. Ich suche vergeblich den entsprechenden Button; erst nach langem Rumprobieren finde ich heraus, dass der Weg zur Wunschliste über den Warenkorb führt.

Also: jeden der drei Artikel in den Warenkorb stellen und von dort aus in die Wunschliste. Effizienz ist was anderes! Aber der Onlineshop hat noch ein paar Macken:

- Das Passwort für das Login zum Onlineshop ist nicht im Firefox-Passwortmanager abrufbar, obwohl ich es dort gespeichert habe. Ist das aufgabenangemessen?

- Nachdem ich dann doch mein Passwort manuell erfasst habe und angemeldet bin, kann ich mich nicht so einfach wieder abmelden, denn der Button zum <Abmelden> befindet sich nicht auf jeder Webseite. Nein, den gibt es nur auf der Seite „Mein Konto“ und die muss ich erst mit der Funktion <Anmelden/Registrieren> aufrufen. Da musst Du drauf kommen!
- Und um das noch zu toppen ist die Funktion <Anmelden/Registrieren> auf jeder Seite am unteren Ende… und die Seiten sind gaaanz laaaang. Das will „erscrollt“ sein!

☹ Zufriedenheit strahle ich nach diesem Einkaufsvorgang nicht aus…

2.2.2 Spezielle Qualitätsmerkmale

Die bisher beschriebenen Qualitätsmerkmale müssen für jede Applikation getestet werden, sei es im Großrechner-, im Client-/Server-, Web- oder Mobile-Umfeld. Für webbasierte und mobile Anwendungen kommen zusätzliche Qualitätsmerkmale zum Tragen.

Eine Website muss den zusätzlichen Anforderungen der Auffindbarkeit, Barrierefreiheit und Rechtskonformität genügen. Diese drei Qualitätsmerkmale werden der in der DIN 66272 [DIN 66272] geforderten Benutzbarkeit zugeordnet und dementsprechend durch die im Kap. 10 beschriebenen Testarten geprüft.

Für eine mobile Anwendung, sei es eine Mobile-Webseite oder eine Mobile-App, müssen die beiden spezifischen Qualitätsmerkmale Mobiltauglichkeit und Mobilfähigkeit eingefordert werden.

Auffindbarkeit Eine Website muss leicht auffindbar sein. Das bedeutet zum einen, dass die Webadresse einen sprechenden, leicht zu merkenden Namen hat und zum anderen, dass sie von Suchmaschinen unter den Top-Treffern angezeigt wird.

Der Auffindbarkeitstest (s. Kap. 10.5) stellt die Auffindbarkeit einer Website sicher.

Barrierefreiheit (Accessibility) Soll eine Webanwendung auch für behinderte Personen gebrauchstauglich sein, so muss sie barrierefrei realisiert werden. Barrierefreiheit nach § 4 des Behindertengleichstellungsgesetzes (BGG) ist wie folgt definiert (s. [URL: BGG]):

„Barrierefrei sind bauliche und sonstige Anlagen, Verkehrsmittel, technische Gebrauchsgegenstände, Systeme der Informationsverarbeitung, akustische und visuelle Informationsquellen und Kommunikationseinrichtungen sowie andere gestaltete Lebensbereiche, wenn sie für behinderte Menschen in der allgemein üblichen Weise, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe zugänglich und nutzbar sind.“

Barrierefreiheit (Accessibility) entspricht somit dem in der ISO 9241 definierten Qualitätsmerkmal Gebrauchstauglichkeit (Usability) für einen Benutzerkreis, der behinderte Menschen mit einschließt.

Diese spezielle Ausprägung des geforderten Qualitätsmerkmals der Benutzbarkeit einer Website wird im Zugänglichkeitstest (s. Kap. 10.4) geprüft.

Rechtskonformität Der Webauftritt muss Rechtens sein, d. h. die weltweit einsehbaren Inhalte von Websites müssen rechtlich „wasserdicht" sein. Eine Website darf zum Beispiel nicht gegen Urheberrechte verstoßen oder verbotene Links einsetzen. Eine Website ist rechtskonform, wenn die dargestellten und verlinkten Inhalte gegen keine gesetzlichen Vorschriften verstoßen.

Die Rechtskonformität einer Website wird im Content-Test (s. Kap. 10.1) geprüft.

Mobiltauglichkeit Eine Website, die auf mobilen Endgeräten aufgerufen werden können soll, oder eine Web-App, die auch auf Smartphones funktionieren soll, muss für diese mobilen Geräte grundsätzlich tauglich sein. Wenn die Darstellung auf einem kleinen Display nicht lesbar ist oder wenn die Bedienelemente nicht bedienbar sind, ist die beste, funktional korrekte Anwendung nichts wert.

Die Prüfung auf Mobiltauglichkeit ist ein Thema beim Mobile-Funktionstest (s. Kap. 9.12).

Mobilfähigkeit Wenn eine Web- oder Mobile-App nicht nur offline benutzt wird, sondern via Funknetzwerk oder WLAN mit einem Back-End-System Daten austauscht, müssen alle kommunizierenden Systemkomponenten optimal entworfen sein und effektiv zusammenarbeiten. Das gesamte System muss mobilfähig sein. Tests müssen zum Beispiel prüfen, wie das System auf dem Smartphone reagiert, wenn der Benutzer mit seiner App Funkzellen-Hopping betreibt.

Mobilfähigkeit ist in mehreren Testarten ein Thema, hat aber besondere Schwerpunkte im Cross-Device-Test (s. Abschn. 9.11.2), im Mobile-Performanz-/Lasttest (s. Kap. 12.7) und im Mobile-Zuverlässigkeitstest (s. Kap. 13.3).

2.3 Wie kann Qualität gemessen werden?

In den beiden vorherigen Abschnitten sind die Qualitätsmerkmale von Software beschrieben. Aber ohne konkrete Maßangaben kann die Erfüllung eines Qualitätsmerkmals wie zum Beispiel Benutzerfreundlichkeit oder Performanz weder positiv noch negativ entschieden werden.

Exkurs: Qualitätsmerkmale eines KFZs

Das Problem ist vergleichbar mit der Anforderung, dass ein neues Auto, das Sie kaufen wollen, wenig Sprit verbrauchen soll. Diese Aussage wird Ihnen jeder Autoverkäufer sofort als besonderes Qualitätsmerkmal aller seiner Fahrzeuge bestätigen. Aber sie beschreibt nur ein subjektives Merkmal des Fahrzeugs, aus der keine Konsequenzen oder Maßnahmen abgeleitet werden können.

Erst die konkrete Vorgabe, dass ein Auto im Durchschnitt höchstens 4,5 l Kraftstoff auf 100 gefahrenen Kilometern verbrauchen darf, hilft weiter. Aus dem Merkmal „geringer Spritverbrauch“ wird so eine messbare Anforderung, deren Nichterfüllung/Erfüllung durch eine konkrete Verbrauchsmessung nachgewiesen werden kann.

In der Software-Entwicklung verhält es sich so, wie bei dem Auto mit geringem Spritverbrauch: Anforderung müssen so formuliert sein, dass sie messbar sind!

Was hilft dem Tester die Forderung, dass die Anwendung performant sein soll?

Eine qualifizierte Aussage zum Qualitätsmerkmal Performanz kann erst mit einer Maßangabe getroffen werden, wie zum Beispiel: „Auf der Website werden Datenblätter für Interessenten zum Download angeboten. Der Download eines Datenblattes darf mit einer DSL-3000-Verbindung nicht länger als 7 s dauern.“

Diese Aussage stellt eine messbare Qualitätsanforderung dar, deren Abweichung oder – hoffentlich – Nichtabweichung durch einen Test konkret in Sekunden angegeben werden kann.

► Eine Qualitätsanforderung besteht aus Qualitätsmerkmal und Qualitätsmaß!

In Software-Entwicklungsprojekten ist es nicht beliebt, Qualitätsmerkmale eines Anwendungssystems mit konkreten Maßangaben zu belegen. Zum einen bedeutet es Aufwand, sich frühzeitig über fachliche und technische Anforderungen detaillierte Gedanken zu machen und dann auch noch festzuschreiben, wie sie zu messen sind. Zum anderen sind dokumentierte Qualitätsanforderungen verbindliche Entscheidungen, mit denen Auftraggeber und Projektleitung in die Verantwortung genommen werden können.

Spätestens zu dem Zeitpunkt, an dem ein IT-Projekt in Terminnöte gerät – und welches IT-Projekt tut das nicht? –, wird das Management versuchen, beim Testen Aufwand zu sparen. Und weniger Tests können bedeuten, dass die ursprünglich gestellten Qualitätsanforderungen nicht eingehalten werden können. Schlechtere Qualität wird geliefert. In solchen Fällen müssen die durch die Sparmaßnahmen entstehenden Konsequenzen und Risiken (s. Kap. 6, Risikoanalyse) bedacht werden. Das gelingt aber nur, wenn zum Projektbeginn die Anforderungen an das zu realisierende Anwendungssystem detailliert und verbindlich festgelegt worden sind.

Aus diesem Grund und um die Qualität einer Anwendung nachweisen zu können, muss der Auftraggeber im Projektauftrag seine Qualitätsanforderungen auch – oder gerade besonders – für alle nicht-funktionalen Qualitätsmerkmale eindeutig festlegen.

► Der Auftraggeber legt die Qualitätsanforderungen fest.

Nicht-funktionale Qualitätsmerkmale sind diejenigen Merkmale, die sich nicht direkt auf die geforderte Funktionalität beziehen. Dazu gehören Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit sowie Auffindbarkeit, Barrierefreiheit und Rechtskonformität für Webanwendungen.

Bei der Festlegung der Qualitätsanforderungen an eine Software muss berücksichtigt werden, dass sich Qualitätsmerkmale widersprechen können. Zum Beispiel erfordert hohe Benutzerfreundlichkeit in der Regel eine Vermehrung des Programmcodes und mindert damit die Effizienz. Hohe Effizienz geht zu Lasten der Änderbarkeit und der Übertragbarkeit eines Programms, weil der Quellcode optimiert werden muss und somit nicht mehr strukturiert und einfach lesbar ist. Aus diesem Grund müssen die Qualitätsmerkmale vom Auftraggeber priorisiert werden.

2.4 Wie kann Qualität erzeugt werden?

In der Qualitätssicherung (QS) werden konstruktive und analytische Qualitätssicherungsmaßnahmen unterschieden.

Weil Qualität nicht im Nachhinein in ein Produkt hinein geprüft werden kann, ist es wichtig, Fehler durch konstruktive Qualitätssicherungsmaßnahmen zu vermeiden. **Konstruktive Qualitätssicherungsmaßnahmen** sind zum Beispiel:

- Auswahl, Einführung und Einsatz von Testmethoden
- Auswahl, Einführung und Einsatz von Testwerkzeugen
- Schulung[8] der handelnden Personen
- Vorgabe von Standards und Beispielen
- Vorgabe verbindlicher Checklisten

Konstruktive Qualitätssicherungsmaßnahmen werden in den frühen Projektphasen vor der Erstellung der Arbeitsergebnisse eingesetzt.

Im Gegensatz dazu greifen **analytische Qualitätssicherungsmaßnahmen**, wenn die zu prüfenden Objekte fertiggestellt sind. Durch sie werden Fehler aufgedeckt und die Qualität der Prüfgegenstände bewertet. Prüfungen von Dokumenten und Tests von Software sind analytische Qualitätssicherungsmaßnahmen.

► Fehler zu vermeiden ist effizienter, als Fehler zu finden.

Die letzte der vier Eingangsfragen, wie Qualität für Software-Anwendungen erzeugt werden kann, lässt sich nur mit dem Wissen um die konstruktiven und analytischen Qualitätssicherungsmaßnahmen beantworten. Dieses Wissen ist die Essenz dieses Buches und wird in den folgenden Kapiteln vermittelt. Die erschöpfende Antwort auf die vierte Qualitätsfrage kann der Leser also nach dem Durcharbeiten dieses Buches geben.

Aber Achtung! Nicht die Qualitätssicherung mit ihren QS-Maßnahmen erzeugt die Qualität einer Software, sondern die Entwicklung. Die Qualitätssicherung unterstützt prä-

[8] Womit das Lesen dieses Buches zu den konstruktiven Qualitätssicherungsmaßnahmen gezählt werden kann. ☺.

ventiv die Entwicklung mit konstruktiven QS-Maßnahmen und weist die geforderte Qualität einer Software mit analytischen QS-Maßnahmen nach.

► Qualität kann nicht in Software hinein getestet werden!

2.5 Zusammenfassung

- Typen von Testobjekten in dialogorientierten Anwendungssystemen sind Web-Apps, Mobile-Apps, Desktop-Apps, Client-Server-Apps, Websites, Mobile- und Desktop-Webseiten.
- Eine Mobile-App kann eine Mobile-Web-App, eine Native- oder eine Hybrid-App sein.
- Die Normen ISO/IEC 25000 und DIN EN ISO 9241 (insbesondere die Teile 10 und 110) definieren die Qualität von Software anhand von Merkmalen.
- Die Hauptqualitätsmerkmale von Software sind Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit und Übertragbarkeit.
- Von einer dialogorientierten Anwendung wird gefordert, dass sie aufgabenangemessen, erwartungskonform, fehlertolerant individualisierbar, lernförderlich, selbstbeschreibungsfähig und steuerbar ist.
- Gebrauchstauglichkeit bedeutet, dass der Benutzer effektiv und effizient mit einer Anwendung arbeiten kann und mit dem „Gesamtkunstwerk“ zufrieden ist.
- Webauftritte bedürfen aufgrund ihrer unbekannten und großen Nutzergemeinde besonderer Anstrengungen, um die zusätzlichen webspezifischen Qualitätssicherungsmerkmale Auffindbarkeit, Barrierefreiheit und Rechtskonformität nachweisen zu können.
- Anwendungen für mobile Endgeräte müssen die Qualitätsmerkmale Mobiltauglichkeit und Mobilfähigkeit erfüllen.
- Erst wenn jedem Qualitätsmerkmal eine entsprechende Messlatte zugeordnet ist, sind die Qualitätsanforderungen an einen Webauftritt bzw. eine Applikation definiert, und erst dann kann die Qualitätssicherung die Abweichungen vom Soll messen.
- Die Qualitätssicherung muss mit geeigneten konstruktiven Qualitätssicherungsmaßnahmen dafür Sorge tragen, dass Fehler vermieden werden.
- Die Qualitätssicherung muss mit angemessenen analytischen Qualitätssicherungsmaßnahmen nachweisen, dass die Qualitätsanforderungen an eine Software erfüllt oder nicht erfüllt werden.

Begriffe zum Testen

3

Wer testet, ist feige.

Zitat eines unverbesserlichen Entwicklers (will anonym bleiben)

In diesem Kapitel werden zuerst die in diesem Buch verwendeten Begriffe zum Testen definiert und erläutert, um die Basis für die folgenden Kapitel zu legen. Das ist notwendig, weil erfahrungsgemäß die Begriffe rund ums Testen von Unternehmen zu Unternehmen, von Projekt zu Projekt, von Tester zu Tester unterschiedlich interpretiert werden.

Anschließend werden die Box-Tests beschrieben, die mit unterschiedlichen Testzielen auf die Testobjekte losgelassen werden.

3.1 Definitionen zum Testen

Die folgenden Definitionen zum Testen orientieren sich zum großen Teil am Glossar des International Software Testing Qualification Boards [ISTQB_2013].

Prüfung ist der Oberbegriff für alle analytischen Qualitätssicherungsmaßnahmen unabhängig von Methode und Prüfgegenstand.

Testen ist der Prozess, der sämtliche (Test-)Aktivitäten umfasst, welche dem Ziel dienen, für ein Software-Produkt die korrekte und vollständige Umsetzung der Anforderungen sowie das Erreichen der festgelegten Qualitätsanforderungen nachzuweisen.

Zum Testen gehören in diesem Sinne auch Aktivitäten zur Planung, Steuerung, Vorbereitung und Bewertung, welche der Erreichung der genannten Ziele dienen.

Ein **Fehler** ist die Nichterfüllung einer festgelegten Forderung. In der Software-Entwicklung fasst ein Fehler die drei Begriffe →Fehlhandlung, →Fehlerzustand und →Fehlerwirkung zusammen.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_3

Eine **Fehlhandlung** (Error) ist die menschliche Handlung, die zu einem →Fehlerzustand in einer Software führt.

Ein **Fehlerzustand** (Fault) ist der Zustand eines Software-Produkts oder einer Software-Komponente, der unter spezifischen Bedingungen eine geforderte Funktion des Software-Produkts beeinträchtigen und zu einer →Fehlerwirkung führen kann. Ein Fehlerzustand wird auch als Defekt, innerer Fehler oder Fehlerursache bezeichnet.

Eine **Fehlerwirkung** (Failure) ist die Wirkung eines Fehlerzustands, die bei der Ausführung der Software/des →Testobjekts für den Benutzer/Tester sichtbar in Erscheinung tritt. Eine Fehlerwirkung wird auch als äußerer Fehler bezeichnet.

Eine **Fehlermaskierung** ist ein Umstand, bei der ein Fehlerzustand oder mehrere Fehlerzustände die Aufdeckung eines anderen Fehlerzustandes verhindern oder seine Fehlerwirkung kompensieren.

Ein **Test** ist eine Aktivität, bei der ein →Testobjekt mittels geeigneter →Testmethoden →statisch überprüft oder →dynamisch ausgeführt wird, mit dem Ziel, Fehler aufzudecken und die korrekte und vollständige Umsetzung der →funktionalen und →nicht-funktionalen Anforderungen nachzuweisen.

Nach [IEEE 829] ist ein Test nicht nur als Aktivität, sondern auch als eine Menge von einem oder mehreren Testfällen definiert.

Ein **Testobjekt** ist ein Dokument, eine Klasse, ein Programm, eine Komponente, ein Teilsystem oder das gesamte System, das einem Test unterzogen wird.

Ein **statischer Test** ist die Prüfung eines Testobjektes ohne Rechnerunterstützung.

Ein **dynamischer Test** ist die Prüfung eines Testobjektes mit Rechnerunterstützung, d. h. unter Ausführung der Software.

Funktionale Anforderungen beschreiben das korrekte Ein- und Ausgabeverhalten und die Verarbeitung der Daten für eine Software-Komponente.

Nicht-funktionale Anforderungen beschreiben das allgemeine Verhalten einer Software-Komponente, wie zum Beispiel die Performanz oder Benutzerfreundlichkeit.

Ein **funktionaler Test** ist ein Test, der die vollständige und korrekte Umsetzung der funktionalen Anforderungen an ein Testobjekt prüft. Die Testfälle für den funktionalen Test werden aus den funktionalen Anforderungen hergeleitet.

Ein **nicht-funktionaler Test** ist ein Test, der die Einhaltung der nicht-funktionalen Anforderungen an ein Testobjekt prüft. Die Testfälle für den nicht-funktionalen Test werden aus den nicht-funktionalen Anforderungen hergeleitet.

Eine **Testart** (Testtyp) ist eine Gruppe von →Testmethoden und Testaktivitäten, die gemeinsam ausgeführt und verwaltet werden, mit dem Ziel, ein Testobjekt auf spezielle Qualitätsmerkmale hin zu überprüfen. Eine Testart kann in mehreren →Teststufen zum Einsatz kommen.[1]

Eine **Teststufe** (Testphase) fasst diejenigen Testaufgaben in einem Software-Entwicklungsprojekt zusammen, die ein bestimmtes →Testziel verfolgen und die gemeinsam

[1] Die einzelnen Testarten sind Inhalt des zweiten Teils dieses Buches.

ausgeführt und verwaltet werden. Teststufen sind mit Zuständigkeiten in einem Projekt verknüpft. Teststufen nach dem V-Modell XT[GL] sind Komponententest, Integrationstest, Systemtest und Abnahmetest.

Teststufen und Testarten werden oftmals gleich benannt (wie zum Beispiel Komponententest, Integrationstest oder Systemtest), obwohl sie per Definition unterschiedliche Bedeutungen haben. Testarten beschreiben Verfahren zur Überprüfung bestimmter Qualitätsmerkmale und können in mehreren Teststufen angewendet werden. Teststufen beschreiben Testaktivitäten, die ein bestimmtes Testziel erreichen sollen und dazu eventuell mehrere Testarten einsetzen.

Ein **Testziel** ist der Grund oder der Zweck für den Entwurf und die Ausführung von Tests.

Eine **Test-Charta** ist eine Anweisung von Testzielen und möglichen Testideen nach denen getestet werden soll.

Eine **Testmethode** ist ein regelwerkbasiertes, systematisches Vorgehen zur Herleitung oder Auswahl von →Testfällen und →Testszenarien, zur Erstellung von →Testspezifikationen und zur Durchführung von Tests.

Ein **Testfall** umfasst die für die Ausführung notwendigen *Vorbedingungen*, die Menge der *Eingabewerte* (ein Eingabewert je Parameter des Testobjekts), die Menge der *vorausgesagten Ergebnisse* und die *erwarteten Nachbedingungen*.

Testfälle werden im Hinblick auf ein bestimmtes Ziel bzw. auf eine →Testbedingung entwickelt, wie zum Beispiel einen bestimmten Programmpfad auszuführen oder die Übereinstimmung mit spezifischen Anforderungen zu prüfen.

Eine **Testbedingung** ist eine Einheit oder ein Ereignis, zum Beispiel eine Funktion, eine Transaktion, ein Feature, ein Qualitätsmerkmal oder ein strukturelles Element einer Komponente oder eines Systems, welche bzw. welches durch einen oder mehrere Testfälle verifiziert werden kann.

Ein **abstrakter Testfall** ist ein Testfall ohne konkrete Ein- und Ausgabewerte für Eingabedaten und vorausgesagte Ergebnisse. Er verwendet abstrakte Operanden, weil die konkreten Werte noch nicht definiert oder verfügbar sind.

Beispiel abstrakter Testfall

Gebe eine in der Datenbank existierende Kundennummer ein; die Kundenadresse wird daraufhin angezeigt.

Ein **konkreter Testfall** ist ein Testfall mit konkreten Werten für Eingaben und vorausgesagten Ergebnissen. Logische Operanden der abstrakten Testfälle werden durch konkrete Werte ersetzt.

Beispiel konkreter Testfall

Gebe die Kundennummer 123456789 ein; die Kundenadresse (mit Name=Xyz,…) wird daraufhin angezeigt.

Eine **Testspezifikation** ist ein Dokument; es beschreibt und begründet die Testfälle und Testszenarien, die in der Testdurchführung zu berücksichtigen sind, und legt die zur Testdurchführung notwendigen Aktivitäten fest.

Ein **Testszenario** (Testablaufspezifikation) ist ein Dokument, das eine Folge von Schritten zur Testausführung festlegt. Auch bekannt als Testskript oder Testdrehbuch [nach IEEE 829]. Es umfasst mehrere, fachlich oder technisch zusammengehörende Testfälle, die gemeinsam in einer vorgegebenen Reihenfolge in einem →Testlauf durchgeführt werden.

Ein **Testlauf** ist die Ausführung eines Testszenarios oder mehrerer Testszenarien mit einer definierten Version eines bestimmten Testobjekts.

Eine **Metrik** (Maß) beschreibt allgemein eine Messvorschrift, um Daten über eine zu untersuchende Einheit zu einer bestimmten Eigenschaft erheben zu können.

Eine **Software-Metrik** ist eine Größe zur Messung eines Qualitätsmerkmals für ein Testobjekt.

Eine **Testmetrik** ist die messbare Eigenschaft eines Testfalls oder Testlaufs mit Angabe der zugehörigen Messvorschrift, wie zum Beispiel eines Testüberdeckungsgrades[2]. Durch Testmetriken werden →Testendekriterien definiert.

Testendekriterien legen fest, unter welchen Voraussetzungen die Tests zu einem Testobjekt eingestellt werden.

Zum Beispiel wird der Test einer Komponente beendet, wenn 100 % aller festgelegten Testfälle und 80 % aller Programmanweisungen ausgeführt worden sind.

Ein **Testkonzept** ist ein Dokument. Es ist die Niederschrift des Testplanungsprozesses für ein Projekt. Im Testkonzept werden u. a.

- der Gültigkeitsbereich, die Vorgehensweise, die Ressourcen und die Zeitplanung der beabsichtigten Tests mit allen Aktivitäten beschrieben,
- die Testobjekte, die zu testenden Features und die Testaufgaben festgelegt,
- die Tester den Testaufgaben zugeordnet und der Unabhängigkeitsgrad der Tester festgelegt,
- die Testumgebung, die Testentwurfsverfahren und die anzuwendenden Verfahren zur Messung der Tests festgelegt und deren Auswahl begründet und
- Risiken beschrieben, die eine Planung für den Fall ihres Eintretens erfordern.

3.2 Box-Tests

Beim Testen gibt es unterschiedliche Sichten auf bzw. in das Testobjekt, die durch die Box-Tests (Blackbox, Whitebox, Greybox) beschrieben werden. Gelegentlich werden die Box-Tests mit den Box-Verfahren verwechselt, weil die gleichnamigen Tests und Verfahren in engem Bezug zueinander stehen. Der folgende Abschnitt erläutert die Unterschiede.

[2] Testüberdeckungsgrade sind im Kap. 4.2.1 beschrieben.

3.2.1 Blackbox-Test

► Ein **Blackbox-Test** ist ein funktionaler oder nicht-funktionaler Test, der ohne Nutzung von Informationen über Interna des Testobjektes ausgeführt wird.

Im Blackbox-Test wird die fehlerfreie und vollständige Umsetzung einer Anforderung überprüft. Dazu werden Tests durchgeführt, ohne dass die internen Strukturen der Programme betrachtet werden, d. h. der Tester sieht ein Testobjekt als Blackbox. Beim Blackbox-Test werden nur die Eingaben und Ausgaben des Testobjektes analysiert. In jeder Teststufe können Tests als Blackbox-Test durchgeführt werden.

Wenn von Blackbox-Tests gesprochen wird, sind oft die im Kap. 4.1 beschriebenen Blackbox-Verfahren zur systematischen Erstellung von Testfällen gemeint. Die beiden Begriffe dürfen aber nicht verwechselt werden. Blackbox-Verfahren sind Testmethoden zur Erstellung von Testfällen, welche anschließend in einem Blackbox-Test ausgeführt werden können.

3.2.2 Whitebox-Test

► Ein **Whitebox-Test** prüft die innere Struktur eines Testobjektes auf Korrektheit und Vollständigkeit, d. h. Systemstrukturen und Programmabläufe sind bekannt und der Programmcode wird analysiert.

Der Whitebox-Test wird daher auch **Strukturtest** oder strukturbasierter Test genannt.

Der Begriff Whitebox-Test wird oft synonym zu den Whitebox-Verfahren verwendet, was nicht richtig ist. Whitebox-Tests sagen etwas über die Art der Testdurchführung aus und können in jeder Teststufe eingesetzt werden. Whitebox-Verfahren dienen der Ermittlung von Testfällen, die in Whitebox-Tests ausgeführt werden können. Whitebox-Verfahren werden im Kap. 4.2 beschrieben.

3.2.3 Greybox-Test

Man ahnt es schon. Der Greybox-Test vereint die beiden Sichtweisen des Blackbox- und des Whitebox-Tests auf das zu testende Objekt. Ein Greybox-Test ist weder eine Testphase, noch liefert er eigene Verfahren zur Ermittlung von Testfällen.

Eine **Greybox** ist eine Komponente, die über ihre Schnittstellen nach außen definiert wird und dessen innere Struktur trotzdem beobachtet werden kann.

► Der **Greybox-Test** prüft eine Greybox auf die richtige Umsetzung der Anforderungen, den korrekten Ablauf und den Aufbau der inneren Strukturen.

In welchen Fällen wird ein Greybox-Test durchgeführt? Klassische Greybox-Tests sind zum Beispiel Integrationstests (s. Kap. 9.3), bei denen bereits produktive und getestete Komponenten zu einem Gesamtsystem zusammengeführt werden, oder Tests für Standardsoftware, die in ein System eingebunden wird.

Der Schwerpunkt der Tests liegt hier auf den Schnittstellen, für die nicht das komplette Testgeschütz aufgefahren werden muss. Die benötigten Tests werden unter Berücksichtigung der Kommunikationsstrukturen der Schnittstellen entworfen und durchgeführt. Der Tester bewegt sich hier im Graubereich zwischen Black- und Whitebox-Tests; die zu integrierenden Testobjekte werden eigentlich als Blackbox betrachtet, aber gleichzeitig muss auf die internen Zusammenhänge geachtet werden.

Da der Greybox-Test keine eigenen Methoden verpflichtet, können ihm weder spezifische Tools noch eigene Qualitätsanforderungen zugeordnet werden – grau eben.

3.3 Zusammenfassung

- Das Standardglossar des ISTQB®/GTB (International Software Testing Qualification Board/German Testing Board e. V.) [ISTQB_2013] legt Testbegriffe auf internationaler Ebene fest.
- Testarten beschreiben Verfahren zur Überprüfung bestimmter Qualitätsmerkmale und können in mehreren Teststufen angewendet werden.
- Teststufen beschreiben Testaktivitäten, die ein bestimmtes Testziel erreichen sollen und dazu eventuell mehrere Testarten einsetzen.
- Blackbox-Tests, Whitebox-Tests und Greybox-Tests beschreiben unterschiedliche „Erleuchtungsgrade“ eines Testobjekts zur Testdurchführung, d. h. die interne Programmstruktur wird beim Test nicht (schwarz), ein wenig (grau) oder vollständig (weiß) ausgeleuchtet.
- Die Blackbox- und Whitebox-Verfahren sind Methoden zur systematischen Erstellung von Testfällen.
- Es gibt kein Greybox-Verfahren zu Ermittlung von Testfällen. Der Greybox-Test vereint die beiden Sichtweisen des Blackbox- und des Whitebox-Tests auf das zu testende Objekt.

Testfallentwurfsverfahren

4

Die Mittelmäßigen klopfen sich zu dem Zeitpunkt auf die Schulter, wo die Könner anfangen zu arbeiten.

Matthias Scharlach (*1947)

Die Kunst des Testens besteht darin, die „richtigen" Testfälle zu finden und zu spezifizieren. „Richtig" in dem Sinne, dass nicht unnötig – sprich unbezahlbar – viele Tests durchgeführt werden, aber trotzdem eine hohe Qualität des Software-Produktes gewährleistet ist.

► Die Anforderung an den Tester lautet: „Teste gut und günstig".

Ein probates Mittel, dieses Ziel zu erreichen, sind Testfallentwurfsverfahren. Sie sorgen dafür, dass Tests möglichst begrenzt in der Anzahl, aber umfassend in der Überdeckung der Testziele sind.

Die im Folgenden beschriebenen Blackbox- und Whitebox-Verfahren zum systematischen Testfallentwurf werden schon viele Jahre praktiziert. Sie sind keine Erfindung des Web- oder Mobile-App-Testings, liefern aber Testfälle für viele der im Teil II beschriebenen Testarten. Jeder, der Testfälle entwirft, muss diese Methoden beherrschen. Ergänzt werden diese beiden Vorgehensweisen durch erfahrungsbasierte Testverfahren.

In diesem Kap. 4 werden nicht alle bekannten Testfallentwurfsmethoden vorgestellt, sondern die, die m. E. für die Tests von Webseiten, Web-Apps und Mobile-Apps besonders effizient eingesetzt werden können. Wer sich für weitere Testmethoden interessiert, möge die entsprechende Literatur studieren, wie zum Beispiel [Bath_2011], [Liggesmeyer_2009] oder [Spillner_2012].

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_4

4.1 Blackbox-Verfahren

Ein erschöpfender Test, der alle möglichen Ein- und Ausgaben zu einem Programm überprüft, ist nicht durchführbar. Um die Anzahl der Testfälle auf ein sinnvolles, aber hinreichendes Maß zu reduzieren, werden die Blackbox-Verfahren angewendet.

Blackbox-Verfahren sind anforderungsbasierte Methoden zur Testfallerstellung. Das bedeutet, sie analysieren die Anforderungsspezifikationen und leiten daraus Testfälle ab. Weil sie dabei die Testobjekte als schwarze Box betrachten, also nicht die inneren Programmstrukturen, werden sie Blackbox-Verfahren genannt. Dazu gehören u. a. die in diesem Buch betrachteten Äquivalenzklassen-, Grenzwert- und Ursache-Wirkungs-Analyse sowie zustands- und anwendungsfallbasiertes Testen. Das Paarweise Testen ist ebenfalls ein Blackbox-Verfahren, das im separaten Kap. 5 beschrieben wird.

Beispiel Finanzierungsrechner
Am Beispiel eines Rechners zur Autofinanzierung werden die einzelnen Schritte erläutert, die im Idealfall bei der Anwendung der Blackbox-Verfahren durchgeführt werden. Beginnen wir mit der Anforderung:

Anforderung lesen
Zu testen ist das Programmmodul <Finanzierungsrechner>, das die Monatsraten für eine Autofinanzierung berechnet.

Der Gesamtfahrzeugpreis wird dem <Finanzierungsrechner> vom Modul <Autokonfigurator> übergeben, nachdem dort das Wunschfahrzeug zusammengestellt worden ist. Die konkreten Anforderungen an den Finanzierungsrechner sind in der Konzeptionsphase beschrieben und natürlich qualitätsgesichert worden. Sie lauten:

Der Kunde muss eine Anzahlung zwischen 2000 und 10.000 € als ganze Zahl eingeben. Die Vertragslaufzeit wird durch drei Radio-Buttons vorgegeben: 12, 24 (initiale Auswahl) oder 36 Monate. Die Laufzeiten haben unterschiedliche Zinskonditionen. Je länger die Laufzeit ist, desto höher sind die jährlichen Zinsen (2, 3, 4 %), welche für den Betrag der Differenz von Fahrzeugpreis und Anzahlung zu zahlen sind.

Zusätzlich kann in einer Checkbox angekreuzt werden, ob ein Altfahrzeug in Zahlung gegeben werden soll (ist standardmäßig nicht angekreuzt). Falls nicht, wird ein Rabatt von 3 % auf den Kaufpreis gewährt, aber nur, wenn die Laufzeit nicht 36 Monate beträgt.

Ein Preisnachlass von 1000 € wird gewährt, wenn der Fahrzeugpreis über 30.000 € liegt und die Vertragslaufzeit auf 12 Monate festgelegt ist.

4.1.1 Äquivalenzklassenanalyse

Eine **Äquivalenzklasse** ist eine Menge von Eingabewerten, die ein identisches funktionales Verhalten eines Testobjektes auslösen, bzw. eine Menge von Ausgabewerten, die ein gleichartiges Verhalten eines Testobjektes aufzeigen.

► Bei der **Äquivalenzklassenanalyse** wird die Menge der möglichen Testfälle anhand der in den Anforderungsspezifikationen beschriebenen Bedingungen in eine endliche Zahl von äquivalenten Klassen unterteilt.

Für alle Elemente aus einer Eingabeäquivalenzklasse wird angenommen, dass sie bei einer Testausführung dieselbe Wirkung erzielen, d. h. die Ergebnisse äquivalent zueinander sind. Daher genügt es für den Test, pro Äquivalenzklasse nur einen **Repräsentanten** auszuwählen. Zwei Repräsentanten (Testdaten) einer Äquivalenzklasse kommen entweder zu einem gleichen Testergebnis oder decken dieselbe Fehlerwirkung auf. So wird einerseits die Anzahl der möglichen Testfälle systematisch reduziert und andererseits erhält man eine hinreichende Anzahl von Testfällen, um – zumindest aus Sicht der Blackbox-Verfahren – die vollständige und korrekte Umsetzung der Anforderungen nachweisen zu können.

Im Rahmen der Äquivalenzklassenanalyse wird jede in den Anforderungen beschriebene Bedingung in Äquivalenzklassen umgeschrieben. Dabei wird jede Äquivalenzklasse entweder den zulässigen Äquivalenzklassen oder den unzulässigen Äquivalenzklassen zugeordnet. **ZulässigeÄquivalenzklassen** beschreiben den regulären Ablauf des Programms. **Unzulässige Äquivalenzklassen** beschreiben geplante Unterbrechungen und Fehlermeldungen im Ablauf eines Programms.

Nicht zu vergessen sind die Äquivalenzklassen für Ausgabebedingungen, wenn „fachlich sinnvolle“ **Ausgabeäquivalenzklassen** identifiziert werden können.

Äquivalenzklassen definieren

Zu jeder in den Anforderungen gestellten Bedingung werden Äquivalenzklassen gebildet. Dabei bleiben Abhängigkeiten der Bedingungen untereinander erst einmal unberücksichtigt.

Unser Finanzierungsrechner hat die drei Eingabemöglichkeiten <Anzahlung>, <Laufzeit> und <Inzahlungnahme Altfahrzeug>. Dazu werden die entsprechenden Äquivalenzklassen (s. Tab. 4.1) gebildet.[1] Die Äquivalenzklassen werden durchnummeriert und mit einem Präfix versehen, um nachweisen zu können, dass jede Äquivalenzklasse in mindestens einem Testfall geprüft wird. Zulässige Äquivalenzklassen erhalten das Präfix „Z“, unzulässige das Präfix „U“.

[1] Um das Beispiel nicht überzustrapazieren, wird im Folgenden auf Prüfungen wie negative Werte, Dezimalpunkte und Nachkommastellen verzichtet.

Tab. 4.1 Äquivalenzklassen

Bedingung	zulässige Äquivalenzklasse	Nr.	unzulässige Äquivalenzklasse	Nr.
Anzahlung	Ganze Zahl 2.000 bis 10.000	Z1	Keine Eingabe Kleiner als 2.000 Größer als 10.000 Nicht ganze Zahl Nicht numerische Eingabe	U1 U2 U3 U4 U5
Laufzeit	12 24 36	Z2 Z3 Z4	Keine Eingabe	U6
Inzahlungnahme Altfahrzeug	Nein Ja	Z5 Z6		

Zu einer Bedingung müssen nicht immer unbedingt ungültige Äquivalenzklassen existieren. Für das Eingabefeld <Inzahlungnahme Altfahrzeug> gibt es zum Beispiel nur gültige Eingaben.

Abstrakte Testfälle zu den Äquivalenzklassen beschreiben
Im nächsten Schritt werden Testfälle und Testszenarien zu den Äquivalenzklassen nach folgenden Regeln festgelegt:

1. Jede Äquivalenzklasse wird von mindestens einem Testfall abgedeckt. Der Nachweis der Vollständigkeit der Testfälle erfolgt durch die Nummerierungen der Äquivalenzklassen, die in der Testfallbeschreibung referenziert werden (Z1, Z2,…, U1, U2,…).
2. Für zulässige Äquivalenzklassen werden die Testfälle so gebildet, dass sie mit möglichst wenigen Testfällen getestet werden können. Wenn man bedenkt, dass zu den Testfällen auch Testdaten und eventuell automatisierte Testskripte zur Durchführung erstellt werden müssen, kann der Aufwand auf diese Weise minimiert werden.
3. Testfälle, die unzulässige Äquivalenzklassen testen, müssen immer genau einen Repräsentanten aus einer unzulässigen Äquivalenzklassen enthalten und sonst nur Repräsentanten aus zulässigen Äquivalenzklassen. D.h. unzulässige Äquivalenzklassen dürfen nicht in einem Testfall zusammengefasst werden, weil sich Fehlersituationen sonst überlagern könnten (Fehlermaskierung).
4. Für jeden Testfall werden die zu erwartenden Testergebnisse festgelegt und beschrieben.

Aus diesen Regeln ergeben sich für unser Beispiel vorerst drei abstrakte Testfälle für die gültigen und sechs für die ungültigen Äquivalenzklassen (Tab. 4.2).

Mit diesen neun abstrakten Testfällen werden alle Eingaben in den Finanzierungsrechner überprüft. Für die Berechnung der korrekten Leasingraten wird allerdings der Kaufpreis, der vom aufrufenden Modul <Autokonfigurator> an den Finanzierungsrechner übergeben

Tab. 4.2 Testfälle zu Äquivalenzklassen

Nr. Testfall (abstrakt)[a]	Äquivalenzklassen	Erwartetes Ergebnis/ Erwartete Reaktion
1	Z1, Z2, Z5	Monatsrate wird richtig berechnet.
2	Z1, Z3, Z5	Monatsrate wird richtig berechnet.
3	Z1, Z4, Z6	Monatsrate wird richtig berechnet.
10	*U1*, Z2, Z5	Hinweis: Sie haben keine Anzahlung eingegeben.
11	*U2*, Z2, Z5	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.
12	*U3*, Z2, Z5	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.
13	*U4*, Z4, Z6	Eingabe nicht möglich, in dem Feld sind nur ganze Zahlen eingebbar.
14	*U5*, Z4, Z6	Eingabe nicht möglich, in dem Feld sind nur Zahlen eingebbar.
15	*U6*, Z1, Z6	Eingabe nicht möglich, die Laufzeit wird mit Radio-Buttons ausgewählt, Standard ist 24 Monate.

[a] Die Nummerierung ist nicht fortlaufend, weil später eventuell noch Testfälle zu gültigen Äquivalenzklassen ergänzt werden müssen

Tab. 4.3 Äquivalenzklassen zum Kaufpreis

Bedingung	Zulässige Äquivalenzklasse	Nr.	Unzulässige Äquivalenzklasse	Nr.
Kaufpreis	13.000 bis 30.000 größer als 30.000	Z7 Z8		

wird, benötigt. Die in Tab. 4.3 aufgeführten Äquivalenzklassen zum Kaufpreis müssen daher noch bei der Testfallerstellung berücksichtigt werden.

Unzulässige Äquivalenzklassen zum Kaufpreis müssen hier nicht definiert und getestet werden, weil das die Aufgabe der Tests zum Autokonfigurator ist. Für den Test des Finanzierungsrechners können wir also davon ausgehen, dass der Autokonfigurator immer korrekt einen ganzzahligen Kaufpreis von mindestens 13.000 € übergibt. Die in Tab. 4.2 festgelegten Testszenarien werden um die neuen Äquivalenzklassen Z7 und Z8 ergänzt, wie in Tab. 4.4 exemplarisch für die ersten drei Testfälle dargestellt ist.

In unserem Beispiel gibt es die zwei Ausgabeäquivalenzklassen „berechnete Monatsrate“ und „Hinweis“. Beide sind mit den Testfällen zu den Eingabeäquivalenzklassen abgedeckt und andere Ausgabebedingungen sind in den Anforderungen nicht beschrieben. Daher müssen die Ausgabeäquivalenzklassen hier nicht explizit behandelt werden.

Tab. 4.4 Testfälle mit Kaufpreis

Nr. Testfall (abstrakt)	Äquivalenzklassen	Erwartetes Ergebnis/ Erwartete Reaktion
1	Z1, Z2, Z5, *Z7*	Monatsrate wird richtig berechnet
2	Z1, Z3, Z5, *Z8*	Monatsrate wird richtig berechnet
3	Z1, Z4, Z6, *Z7*	Monatsrate wird richtig berechnet

Für den Finanzierungsrechner sind nun alle Äquivalenzklassen mit abstrakten Testfällen abgedeckt. Aber der engagierte Tester möchte natürlich noch prüfen, ob das Programm auch an den Rändern der möglichen Eingaben, d. h. an den Grenzen der Äquivalenzklassen, richtig funktioniert.

4.1.2 Grenzwertanalyse

► Die **Grenzwertanalyse** stellt sicher, dass Fehler in den kritischen Grenzbereichen der Äquivalenzklassen gefunden werden.

Dazu muss jeder Rand einer gültigen bzw. ungültigen Äquivalenzklasse – der **Grenzwert** – mit mindestens einem Testfall abgedeckt werden. Für den Kaufpreis unseres Finanzierungsrechners ergeben sich zum Beispiel die Grenzwerte 12.999, 13.000, 30.000 und 30.001 für die Äquivalenzklassen Z7 und Z8.

In dem Beispiel wurden die **einseitigen Grenzwerte** genommen, d. h. für die Äquivalenzklassen Z7 der Wert an der Grenze selbst (13.000) und der von der angrenzenden Äquivalenzklasse (12.999). Für ganz genaue Tester gibt es den Ansatz der **beidseitigen oder zweifachen Grenzwerte**, bei dem beide Werte rechts und links von dem eigentlichen Grenzwert genommen werden. Für die 13.000 in unserem Beispiel sind das 12.999, 13.000 und 13.001.

Man kann nachweisen, dass diese Methode der doppelten Grenzwerte zusätzliche Fehlerzustände aufdeckt; sie treibt natürlich den Testaufwand in die Höhe, ist aber für kritische Algorithmen sicherlich angebracht. In kaufmännischen Systemen und Web-Auftritten kann m. E. darauf verzichtet werden.

Grenzwerte, die mit Hilfe von Äquivalenzklassen gefunden werden, stellen Repräsentanten im Sinne der Äquivalenzklassenanalyse dar. Sie sollten aber nicht die Repräsentanten aus den Äquivalenzklassen ersetzen, die nicht auf den Grenzen liegen, sondern diese ergänzen.

Tab. 4.5 Grenzwerte zu Äquivalenzklassen

Bedingung	Grenzwert zulässige Äquivalenzklasse	Nr.	Grenzwert unzulässige Äquivalenzklasse	Nr.
Anzahlung	2.000 10.000	Z1-GW1 Z1-GW2	0 1.999 10.001 [Größte eingebbare Zahl]	U2-GW1 U2-GW2 U3-GW1 U3-GW2
Kaufpreis	12.999 13.000 30.000 30.001	Z7-GW1 Z7-GW2 Z8-GW1 Z8-GW2		

Grenzwerte festlegen
Für unser Beispiel bilden wir noch für das Feld <Anzahlung> die einseitigen Grenzwerte zu den Äquivalenzklassen Z1, U2 und U3:

- Z1 hat die Grenzwerte 1999, 2000, 10.000, 10.001
- U2 hat die Grenzwerte −1, 0, 1999, 2000
- U3 hat die Grenzwerte 10.000, 10.001, [Größte eingebbare Zahl], [Größte eingebbare Zahl]+1

Weil wir negative Zahlen hier nicht betrachten wollen (s. o.), einige Grenzwerte doppelt vorkommen und der Wert [Größte eingebbare Zahl]+1 per Definition nicht existiert, ergeben sich für den Finanzierungsrechner die in Tab. 4.5 dargestellten Grenzwerte. Sie werden je Äquivalenzklasse mit „GW“ gekennzeichnet und durchnummeriert.

Testszenarien um Grenzwerte ergänzen
Die bisher beschriebenen Testfälle werden um die Grenzwerttestfälle ergänzt. Das Ergebnis zu unserem Beispiel ist in Tab. 4.6 dargestellt. Die Grenzwerte Z7-GW1 und Z8-GW2 des Kaufpreises müssen an dieser Stelle nicht betrachtet werden, weil beim Test des Autokonfigurators sichergestellt werden muss, dass diese Werte niemals an den Finanzierungsrechner übergeben wird.

Halten wir folgende Erkenntnisse zur Grenzwertanalyse fest:

- Bei der Grenzwertanalyse werden mit den Grenzwerten die konkreten Testdaten für einen Testfall vorgegeben, wie zum Beispiel 2.000 in Z1-GW1.
- Auch die Grenzen an ungültigen Äquivalenzklassen müssen betrachtet werden. Im Beispiel sind das U2 und U3.
- Nicht zu jeder Äquivalenzklasse gibt es sinnvolle Grenzwerte, wie im Beispiel zu den Äquivalenzklassen Z5, Z6, U1, U4, U5, U6.

Tab. 4.6 Testfälle mit Grenzwerten

Nr. Testfall (abstrakt)	Äquivalenzklassen mit Grenzwerten	Erwartetes Ergebnis/Erwartete Reaktion
4	Z1-GW1, Z2, Z5, Z7-GW2	Monatsrate wird richtig berechnet.
5	Z1-GW2, Z2, Z5, Z8-GW1	Monatsrate wird richtig berechnet.
6	Z1, Z2, Z5, *Z8-GW2*	Monatsrate wird richtig berechnet.
16	*U2-GW1*, Z2, Z5, Z7	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.
17	*U2-GW2*, Z2, Z5, Z7	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.
18	*U3-GW1*, Z2, Z5, Z8	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.
19	*U3-GW2*, Z2, Z5, Z8	Hinweis: Die Anzahlung muss zwischen 2.000 und 10.000 € liegen.

- Gültige Äquivalenzklassen können auch Grenzwerte darstellen, wie im Beispiel die Äquivalenzklassen Z2 und Z4, welche die Grenzen zur Eingabe der Laufzeit darstellen.

4.1.3 Ursache-Wirkungs-Analyse

Bisher wurden keine Abhängigkeiten zwischen den Bedingungen der Anforderungen berücksichtigt. Aber gerade die möglichen Kombinationen ergeben die „interessanten" Testfälle, die in der Analyse von Ursache und Wirkung untersucht werden.

► Mit Hilfe der **Ursache-Wirkungs-Analyse** werden Kombinationen von zulässigen Äquivalenzklassen, die unterschiedliche Programmreaktionen zur Folge haben, auf ihr korrektes Zusammenspiel hin getestet.

Weil Tests zu den Testfällen, die auf unzulässigen Äquivalenzklassen basieren, in jedem Fall Fehlermeldungen als Ergebnis liefern und keine korrekte Durchführung der Funktionalitäten zulassen, werden bei der Ursache-Wirkungs-Analyse nur zulässige Äquivalenzklassen betrachtet.

In der Literatur wird für die Testfallerstellung von Eingabekombinationen die Methode der Ursache-Wirkungs-Graphen beschrieben (zum Beispiel bei [Myers_1987][2]). Diese ist eine formale Sprache, die in komplexen Graphen die kombinatorischen Zusammenhänge beschreibt.

[2] Mein erstes Buch zum Testen, daher muss es genannt werden! ☺

Bei der Testfallermittlung kommerzieller Anwendungen haben sich **Entscheidungstabellen** zur Darstellung von Kombinationen möglicher Ursachen und den daraus resultierenden Wirkungen bewährt. Zudem eignen sich Entscheidungstabellen aufgrund ihrer guten Lesbarkeit optimal zur Kommunikation mit Fachabteilungen. Eine Entscheidungstabelle besteht aus vier Blöcken:

Ursachen	Regeln
Wirkungen	Entscheidungen

- **Ursachen**
  sind Situationen oder Bedingungen, die eine Entscheidung beeinflussen. Für die Testfallerstellung sind das die zulässigen Äquivalenzklassen.
- **Wirkungen**
  listen die möglichen Ausgaben, Programmreaktionen oder Systemzustände auf.
- **Regeln**
  beschreiben die Kombinationsmöglichkeiten der Ursachen. Sie werden für die Testfallermittlung aus den Anforderungsbeschreibungen abgeleitet und bilden die fachlichen Abhängigkeiten der zulässigen Äquivalenzklassen untereinander ab.
- **Entscheidungen**
  beschreiben, welche konkreten Wirkungen aufgrund einer Regel erwartet werden.

Entscheidungstabelle mit Abhängigkeiten aufstellen
Die bei Äquivalenzklassen- und Grenzwertanalyse vernachlässigten Abhängigkeiten von Anforderungsbedingungen werden nun explizit analysiert.

Für unseren Finanzierungsrechner ergeben sich aus den beschriebenen Anforderungen 12 Kombinationen für <Laufzeit>, <Inzahlungnahme Altfahrzeug> und <Kaufpreis> (s. Tab. 4.7). Die <Anzahlung> hat keinen Einfluss auf die Entscheidungsfindung und wird daher in der Tab. 4.7 nicht berücksichtigt, was nicht heißt, dass die Höhe der Anzahlung keinen Einfluss auf die zu berechnende Monatsrate hat.

Entscheidungstabelle reduzieren
Entscheidungstabellen können sehr komplex werden. Bei n binären Bedingungen erhält man 2^n Kombinationen und bei mehr als zwei Möglichkeiten zu einer Bedingung noch wesentlich mehr. In der Praxis wird man in solchen Fällen, falls das fachliche Risiko tragbar ist, eine repräsentative Auswahl aus allen Kombinationsmöglichkeiten treffen.

Diese Auswahl kann durch die Fachabteilung geschehen, die festlegt, welche kombinatorischen Testfälle aus fachlicher Sicht besonders wichtig bzw. zu vernachlässigen sind.

Zur Reduzierung der Entscheidungstabelle können auch die Wirkungen betrachtet werden. Wenn für unterschiedliche Regeln die Wirkungen identisch sind, dann werden diese Regeln auf Gemeinsamkeiten und Optimierungsmöglichkeiten untersucht. Bedingungen der Reduzierung sind, dass jede Kombination der Wirkungen mit einem Testfall abgedeckt und jede Ursache mindestens einmal berücksichtigt werden muss.

Tab. 4.7 Testfallkombinationen

Ursachen			*Regeln*											
			1	2	3	4	5	6	7	8	9	10	11	12
Laufzeit	12=2%	Z2	X	X	X	X								
	24=3%	Z3					X	X	X	X				
	3=4%	Z4									X	X	X	X
Inzahlungnahme Altfahrzeug	Nein	Z5	X	X			X	X			X	X		
	Ja	Z6			X	X			X	X			X	X
Kaufpreis	<=30.000	Z7	X		X		X		X		X		X	
	>30.000	Z8		X		X		X		X		X		X
Wirkungen			*Entscheidungen*											
3% Rabatt auf Kaufpreis		W1	J	J	N	N	J	J	N	N	N	N	N	N
Preisnachlass 1.000		W2	N	J	N	J	N	N	N	N	N	N	N	N

Tab. 4.8 Reduzierte Testfallkombinationen

Ursachen			*Regeln*						
			1	2	3	4	6	8	11
Laufzeit	12=2%	Z2	X	X	X	X			
	24=3%	Z3					X	X	
	36=4%	Z4							X
Inzahlungnahme Altfahrzeug	Nein	Z5	X	X			X		
	Ja	Z6			X	X		X	X
Kaufpreis	<=30.000	Z7	X		X				X
	>30.000	Z8		X		X	X	X	
Wirkungen			*Entscheidungen*						
3% Rabatt auf Kaufpreis		W1	J	J	N	N	J	N	N
Preisnachlass 1.000		W2	N	J	N	J	N	N	N

In unserem Beispiel können die Regeln 9–12 zusammengefasst werden, weil bei einer Laufzeit von 36 Monaten kein Rabatt und kein Preisnachlass gewährt werden; Regel 11 wird zufällig als zu testende Regel ausgewählt.

Die Regeln 5 und 6 sowie 7 und 8 werden jeweils zusammengefasst, weil bei 24-monatiger Laufzeit nie ein Preisnachlass gewährt wird und der Rabatt nur von der Inzahlungnahme abhängt. Auf diese Weise bleiben sieben kombinatorische Testfälle, s. Tab. 4.8.

Ein weiteres, systematisches Vorgehen zur Reduzierung der Regeln einer komplexen Entscheidungstabelle ist das **Pairwise-Verfahren**[3]. Dabei werden in einer Entscheidungstabelle die Regeln gestrichen, die doppelte Paare von gültigen Äquivalenzklassen enthal-

[3] Als Mathematikfan habe ich dem Thema „Paarweises Testen“, zu dem das Pairwise-Verfahren gehört, das eigene Kapitel 5 spendiert.

Tab. 4.9 Testfälle Ursache-Wirkung

Nr. Testfall (abstrakt)	Äquivalenzklassen	Erwartetes Ergebnis/ Erwartete Reaktion
7	Z1, Z2, Z6, Z7	Monatsrate wird richtig berechnet
8	Z1, Z2, Z6, Z8	Monatsrate wird richtig berechnet
9	Z1, Z3, Z6, Z8	Monatsrate wird richtig berechnet

ten. Auf diese Weise werden zumindest alle Kombinationen von Äquivalenzklassenpaaren getestet. Bei den Streichungen muss darauf geachtet werden, dass jede mögliche Wirkung mindestens einmal zum Tragen kommt, da die Testüberdeckung sonst nicht vollständig wäre.

Mit dem Pairwise-Verfahren würden sechs kombinatorische Testfälle der Entscheidungstabelle übrig bleiben – probieren Sie es aus, wenn Sie Kap. 5 gelesen haben.

In einem realen Projekt würde eine Tabelle mit 12 Einträgen nicht reduziert werden, weil der Testaufwand für diese Anzahl von Testfällen vertretbar ist. Bei umfangreichen Entscheidungstabellen sollte das Reduzierungsverfahren von einem Werkzeug durchgeführt werden (s. Abschn. 4.1.6). Steht kein Testwerkzeug für die Reduzierung von Entscheidungstabellen zur Verfügung, müssen die Streichungen von Personen aus der Fachabteilung vorgenommen werden. Dazu wird in der Regel nicht das Pairwise-Verfahen angewendet, sondern die Eingabekombinationen werden auf Basis einer Risikobewertung gestrichen.

Testszenarien um Kombinationen ergänzen

Wenn man die Regeln in Tab. 4.8 mit den Testfällen in Tab. 4.4 und 4.6 vergleicht, stellt man fest, dass die Kombination der gültigen Äquivalenzklassen der Regeln 3, 4 und 8 in keinem der Testfälle enthalten ist. Somit ergeben sich zusätzliche Testfälle (Tab. 4.9):

Konkrete Testfälle definieren

Bisher wurden bis auf einige Grenzwerttestfälle nur abstrakte Testfälle beschrieben. Im Rahmen der Testvorbereitung müssen für alle Testfälle die konkreten Werte festgelegt werden, mit denen die Tests durchgeführt werden sollen. Besonders für Berechnungen ist es wichtig, die Eingabewerte und Ergebnisse konkret zu bestimmen. Denn dann kann das Testergebnis vom Tester eindeutig überprüft oder bei automatisierten Tests vom Testwerkzeug mit dem Sollwert verglichen werden.

Für den ersten abstrakten Testfall unseres Finanzierungsrechners werden z. B. folgende konkrete Werte für den Test vorgegeben:

Eingabeparameter	Äquivalenzklasse	Eingabewert
Anzahlung	Z1	4000
Laufzeit	Z2	12 Monate = 2 % Zinsen
Inzahlungnahme Altfahrzeug	Z5	Nein
Kaufpreis	Z7	24.000
Erwartetes Ergebnis/Erwartete Reaktion		1700 € Monatsrate

Mit den bis hier beschriebenen Verfahren zur Testfallermittlung wird normalerweise der größte Teil der durchzuführenden Testfälle für den anforderungsbasierten Test erkannt und beschrieben, d. h. eine hohe Anforderungsüberdeckung durch Testfälle wird erreicht. Anhand von Zustands- und Anwendungsfalldiagrammen können weitere fachlich bedingte Testfälle entworfen werden. Diese Arten der Testfallermittlung werden in den beiden nächsten Abschnitten beschrieben.

4.1.4 Zustandsbasierte Testfallermittlung

Objekte und Systeme können unterschiedliche Zustände annehmen. In der Entwicklung von Realtime-Systemen (wie z. B. Ampelanlagen oder Getränkeautomaten) werden Zustandsübergangsdiagramme (State Transition Diagram) „schon immer“ als Mittel zur Anforderungsbeschreibung von Zustandsänderungen eingesetzt.

Bei der Entwicklung von Webapplikationen sind technische Zustände von Systemen in der Regel nicht relevant, hier durchlaufen fachliche Objekte nach bestimmten Regeln verschiedene Lebensabschnitte. Die Status, die ein Objekt einnehmen kann, und die dazugehörenden Statusübergänge werden in Zustandsdiagrammen dargestellt. **Zustandsdiagramme** – alias Zustandsübergangsdiagramme – sind ein sehr mächtiges und für die Fachabteilung verständliches Beschreibungsmittel. Aus ihnen lassen sich recht einfach Testfälle für einen zustandsbasierten Test entwickeln.

► Die **zustandsbasierte Testfallermittlung** entwirft anhand von Zustandsdiagrammen Testfälle zur Überprüfung der Vollständigkeit, Richtigkeit und Erreichbarkeit der Zustände und Zustandsübergänge eines Testobjekts.

In unserem Beispiel durchläuft das Objekt <Finanzierungsanfrage> unterschiedliche Zustände (Abb. 4.1).

Ein Kunde kann mit dem Finanzierungsrechner eine Finanzierungsmöglichkeit berechnen und abspeichern (Zustand „angelegt“). Natürlich kann er die Berechnung vor dem Speichern verwerfen oder abbrechen, dann ist allerdings kein Zustand der Finanzierungsanfrage erreicht, der im System abgelegt werden müsste.

Zu einem späteren Zeitpunkt kann der Kunde eine angelegte Finanzierungsanfrage verändern (immer noch Zustand „angelegt“), löschen (Zustand „gelöscht“) oder eine konkrete Anfrage starten (Zustand „angefragt“). Vom Händler erhält er darauf eine verbindliche

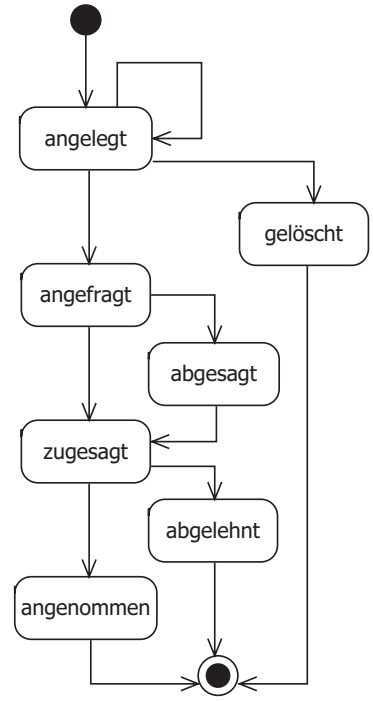


Abb. 4.1 Zustandsdiagramm Finanzierungsanfrage

Zusage (Zustand „zugesagt“) oder eine Absage (Zustand „abgesagt“). Eine Absage kann vom Autohändler später doch noch in eine Zusage umgewandelt werden. Im positiven Fall hat der Kunde zwei Wochen Zeit, diese Zusage anzunehmen, was einen Vertragsabschluss zur Folge hat (Zustand „angenommen“). Lässt er die Zusage verfallen oder lehnt er sie explizit ab, führt das zum Zustand „abgelehnt“.

Der Zustand „gelöscht“ besagt, dass das Objekt seitens Kunde gelöscht wurde und nicht mehr von der Anwendung angezeigt wird, allerdings noch in der Datenbank vorhanden ist. Alle anderen Zustände sind, wenn sie erreicht sind, in der Datenbank abgelegt und jederzeit abrufbar.

Die Abb. 4.1 stellt diese Anforderungen für eine konkrete Finanzierungsanfrage in einem Zustandsdiagramm dar.

Aus Übersichtsgründen sind keine Ereignisse und Funktionen (Methoden) abgebildet, die zu einer vollständigen Beschreibung eines Zustandsdiagramms gehören. In Abb. 4.2 sind exemplarisch die Ereignisse „14 Tage vorbei“ und „Kunde lehnt ab“ mit der durch sie ausgelösten Methode „Ablehnen“ dargestellt. Sie führen den Zustandsübergang einer Finanzierungsanfrage von „zugesagt“ nach „abgelehnt“ herbei.

In einem Zustandsdiagramm sind eine Vielzahl fachlicher Anforderungen enthalten, so auch in Abb. 4.1 und 4.2:

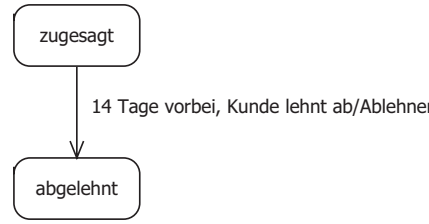


Abb. 4.2 Zustandsdiagramm mit Ereignissen

- Jeder Zustandsübergang wird von mindestens einem Ereignis herbeigeführt und jedes Ereignis muss von der Anwendung bearbeitet werden.
- Zu jedem Ereignis muss eine verarbeitende Funktion (Methode) spezifiziert und realisiert sein. Im Beispiel werden die beiden Ereignisse „14 Tage vorbei“ und „Kunde lehnt ab“ von der Methode <Finanzierungsanfrage.Ablehnen> bearbeitet.
- Ein Zustandsdiagramm legt Reihenfolgen fest, in denen Ereignisse eintreten müssen bzw. eintreten dürfen, um verarbeitet werden zu können. Um im Beispiel eine vom Händler zugesagte Finanzierung annehmen zu können, muss zuvor der Zustand „zugesagt“ erreicht worden sein.
- Nicht erlaubte Aufrufsequenzen von Funktionen einer Anwendung sind sofort ersichtlich. Im Beispiel darf eine Finanzierungsanfrage nur gelöscht werden, wenn sie im Zustand „angelegt“ ist. In jedem anderen Zustand darf die Funktion/Methode zum Löschen des Objektes nicht ausführbar bzw. für den Benutzer in der Anwendung erst gar nicht sichtbar sein.

In unserem Beispiel drängen sich beim Review des Zustandsdiagramms folgende Fragen auf, die fachlich zu klären wären:

- Darf eine zugesagte Finanzierung geändert werden?
- Wann kann ein Finanzierungsangebot von wem storniert werden? Ist „storniert“ ein weiterer Zustand?
- Wer braucht den Zustand „gelöscht“, wenn er keinem Benutzer angezeigt wird? Wann wird eine Anfrage in diesem Zustand physikalisch aus der Datenbank gelöscht?
- Ist die Vertragsabwicklung der angenommenen Finanzierung in weiteren Zuständen dieses Objektes abzubilden (unterschrieben, aktiv, still gelegt, abgeschlossen) oder handelt es sich um ein neues Objekt <Vertrag> mit eigenen Zuständen?

Ich habe sehr positive Erfahrungen mit dieser Spezifikationsmethode gemacht und kann sie aus Sicht der Qualitätssicherung nur empfehlen. In der Projektarbeit entbrennen über dieser Art der Darstellung der fachlichen Zusammenhänge sehr konstruktive Diskussionen, weil schnell Lücken in der Spezifikation aufgedeckt werden.

Zustandsbasierte Testszenarien entwerfen

Aus einem Zustandsdiagramm werden Testfälle abgeleitet, die aufgrund der erlaubten Reihenfolgen zu Testszenarien zusammengeführt werden. Die Menge der zustandsbasierten Testfälle zu einem Zustandsdiagramm müssen folgende Minimalforderungen erfüllen:

- Jeder Zustand wird erreicht.
- Jedes Ereignis wird ausgelöst.
- Jeder erlaubte Zustandsübergang wird ausgeführt.
- Es wird nachgewiesen, dass jeder nicht erlaubte Zustandsübergang nicht ausgeführt werden kann.

Es gibt noch weitere Metriken zur Testfallüberdeckung von Zustandsdiagrammen, wie n-Switch- oder Rundreiseüberdeckung, die hier nicht betrachtet werden.

Zustandsdiagramme können sich über mehrere Anwendungsfälle (s. Abschn. 4.1.5) erstrecken und sogar komplette Geschäftsprozesse abbilden. Daher sind zustandsbasierte Testszenarien nicht nur für die Tests von Objekten im Unit-Test (s. Kap. 9.1) relevant, sondern auch für die Tests von Geschäftsprozessen im funktionalen Systemtest (s. Kap. 9.4).

4.1.5 Anwendungsfallbasierte Testfallermittlung

Eine Möglichkeit, Testfälle auf Basis eines objektorientierten Software-Entwurfs zu erstellen, ist die Analyse von Anwendungsfällen.

► Die **anwendungsfallbasierte Testfallermittlung** entwirft anhand von Anwendungsfalldiagrammen Testszenarien zur Überprüfung der vollständigen und korrekten Umsetzung der spezifizierten Anwendungs- und Geschäftsvorfälle in einem Software-System.

Ein **Anwendungsfall** (Use Case) beschreibt Interaktionen, die zwischen Akteuren und dem betrachteten System stattfinden, um ein bestimmtes fachliches Ziel zu erreichen. Ein Anwendungsfall wird durch einen Akteur angestoßen und führt zu einem konkreten Ergebnis für die handelnden Akteure.

Mehrere voneinander abhängige oder in Beziehung stehende Anwendungsfälle bilden einen Geschäftsprozess und werden in einem **Anwendungsfalldiagramm** dargestellt, deren einzelne Elemente im Rahmen der Anforderungsbeschreibung detailliert spezifiziert werden. Ein Anwendungsfalldiagramm (Use Case Diagram) ist eine Diagrammart der UML[4] und zeigt die Zusammenhänge der Anwendungsfälle mit Akteuren, untereinander und zum Systemumfeld.

Ein Anwendungsfalldiagramm zu unserem Finanzierungsrechner ist in Abb. 4.3 dargestellt. Darin werden die drei Anwendungsfälle „Auto konfigurieren“, „Finanzierungsmöglichkeit berechnen“ und „Finanzierungsanfrage stellen“ mit ihren Beziehungen abgebildet. Beziehungen bestehen zu den beiden Akteuren „Kunde“ und „Händler“ sowie untereinander in einer <<include>> -Beziehung und einer <<extend>>-Beziehung. Die <<include>> -Beziehung sagt aus, dass mit dem Anwendungsfall „Finanzierungsmöglich-

[4] Unified Modeling Language (UML) ist eine standardisierte Sprache zum objektorientierten Software-Engineering, [URL: UML-OMG].

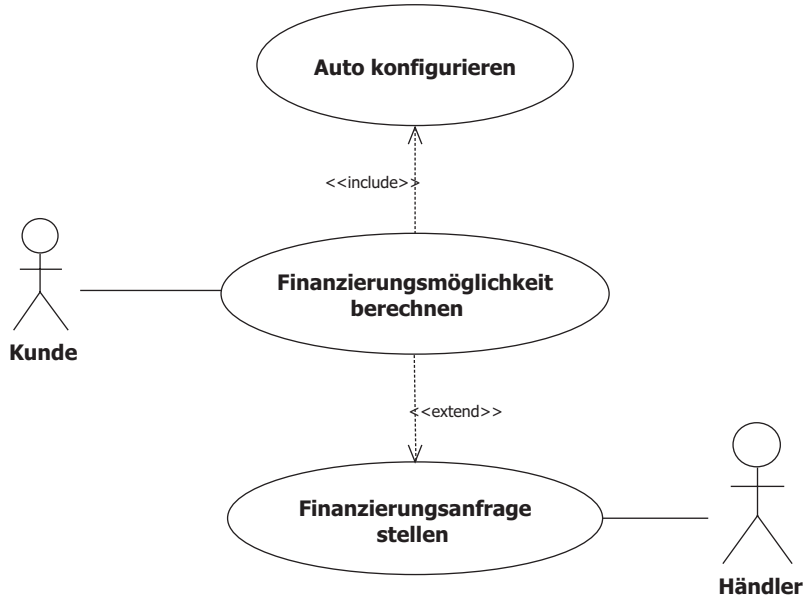


Abb. 4.3 Anwendungsfalldiagramm

keit berechnen“ immer „Auto konfigurieren“ ausgeführt werden muss. Die <<extend>>-Beziehung ist optional, d. h., „Finanzierungsanfrage stellen“ wird nur unter zu spezifizierenden Bedingungen ausgeführt.

Anwendungsfallbasierte Testszenarien ergänzen
Auf Basis der Anwendungsfälle und ihrer Beziehungen können Testfälle entwickelt werden. Allerdings sind nicht alle Informationen, die für die Realisierung und die Testdurchführung notwendig sind, allein aus dem Anwendungsfalldiagramm ersichtlich. Details zu einem Anwendungsfall wie Ausgangsbedingungen, Bedingungen für <<extend>> -Beziehungen oder Nachbedingungen sind in den fachlichen Spezifikationen zu entnehmen, die zu jedem Anwendungsfall beschrieben werden müssen.

Die anwendungsfallbasierten Testszenarien sind besonders für den Systemtest und den Abnahmetest interessant, weil sie die fachlichen Prozesse überprüfen und Systemzusammenhänge darstellen. Im Abschn. 9.4.1 ist ein Beispiel eines anwendungsfallbasierten Testszenarios zum funktionalen Systemtest beschrieben.

4.1.6 Werkzeuge für die Testspezifikation

Die Dokumentation von Testfällen und Testszenarien kann in einem Text- oder Tabellenkalkulationsprogramm erfolgen. Bei komplexen Anwendungen wird es für ein Projekt allerdings sehr schwer, die Testdokumentation einheitlich und aktuell zu halten.

Besonders die Verknüpfungen von einer Äquivalenzklasse zum Testfall, vom Testfall zum Testszenario, vom Testszenario zum Testdatensatz und eventuell vom Testfall zu

einem gefundenen Fehler sind manuell kaum abzubilden. Solche Referenzen werden aber für die Nachvollziehbarkeit der Tests und den Nachweis des Testfallüberdeckungsgrades benötigt.

Weitere Probleme bei der manuellen Testdokumentation liegen darin, umfangreiche Entscheidungstabellen übersichtlich darzustellen, daraus Testszenarien zu generieren und alle Dokumente bei Änderungen konsistent zu halten.

Daher ist es sinnvoll, Testtools einzusetzen, die eine lückenlose Dokumentation ermöglichen und das gesamte Projektteam in seiner Arbeit unterstützen.

Zum methodenunterstützten Testfallentwurf gibt es einige kommerzielle Werkzeuge. Eines davon ist **CaseMaker**, das die Pairwise-Methode unterstützt und unter bestimmten Voraussetzungen aus Zustandsdiagrammen Testfälle generieren kann, [URL: CaseMaker].

Ein empfehlenswertes Open Source Tool zum verwalten von Testfällen ist **TestLink**, [URL: TestLink]. TestLink kann mit diversen Fehlermanagementtools wie z. B. Mantis verbunden werden, so dass Testfälle, Testdurchführung und darauf referenzierte Fehler nachvollziehbar dokumentiert werden können.

Weitere **Tools zur Testspezifikation** sind zum Beispiel bei [URL: TestToolsOPEN-SOURCE] in der Rubrik „Testing Tools – Test Management“ zu finden.

4.1.7 Qualitätsanforderungen zu den Blackbox-Verfahren

Eine Qualitätsanforderung an die Blackbox-Verfahren ist die Anforderungsüberdeckung. Der Grad der **Anforderungsüberdeckung** gibt an, wie viel Prozent der definierten Anforderungen durch Testfälle überprüft werden.

Der **Testfallüberdeckungsgrad** ist ein Qualitätsmaß zur Testdurchführung. Er gibt an, wie viel Prozent der definierten Testfälle nachweislich im Test ausgeführt wurden.

Damit der Testaufwand eines Projektes im budgetierten Rahmen bleibt, sollten Testfallüberdeckungsgrade pro Risikoklasse festgelegt werden. Dazu müssen alle Testobjekte einer Risikoklasse zugeordnet werden. Für jede Risikoklasse wird ein bestimmter, nachzuweisender Testfallüberdeckungsgrad festgelegt, wie zum Beispiel[5]:

- Risikoklasse A: 100 % Testfallüberdeckung
- Risikoklasse B: 80 % Testfallüberdeckung
- Risikoklasse C: 50 % Testfallüberdeckung

Der Nachweis der Erfüllung eines geforderten Testfallüberdeckungsgrades zu einem Testobjekt kann natürlich erst vorgelegt werden, nachdem die Tests mit den beschriebenen Testfällen durchgeführt wurden.

[5] Im Kapitel 6.3 sind in Tab. 6.2 Risikoklassen für Programme definiert, die für das beschriebene Beispiel angewendet werden können.

4.1.8 Empfehlungen zu den Blackbox-Verfahren

Testfälle früh entwerfen
Die Entwicklung von Testfällen nach den Blackbox-Verfahren ist eine sehr effektive Qualitätssicherungsmaßnahme. Beim Testfallentwurf muss jedes Detail zur Eingabe, Verarbeitung und Ausgabe eines Testobjekts vollständig beschrieben werden, so dass fehlerhafte oder fehlende Anforderungen sofort offensichtlich werden. Daher muss es ein Ziel der Qualitätssicherung sein, Testfälle möglichst früh zu entwerfen.

Testfälle genehmigen lassen
Die mit den Blackbox-Verfahren entworfenen Testfälle sollten einem Abnahme-Review[6] durch Vertreter der Fachabteilung und des Entwicklerteams unterzogen werden. Das hat zwei gute Gründe.

Erstens bestätigt die Fachabteilung mit der Abnahme der Testszenarien die fachliche Richtigkeit und Vollständigkeit der geplanten Tests und somit indirekt die der Anforderungskonzepte. Sind die Testfälle falsch, so sind entweder die Anforderungen nicht hinreichend beschrieben oder die Testfälle sind nicht richtig. In beiden Fällen müssen Verbesserungen vorgenommen werden.

Zweitens bestätigt das Entwicklerteam durch die Abnahme der Testszenarien, dass die Anforderungen realisierbar sind und die Testfälle der Realisierung nicht widersprechen. Aufgrund des gemeinsamen Verständnisses der Anforderungen und der geplanten Tests werden Fehler bei der Programmierung vermieden.

Testfälle bewerten
Um sicherzustellen, dass die kritischen Tests zuerst durchgeführt werden, sollten Testfälle und Testszenarien einer Risikoklasse zugeordnet werden. Von Fachleuten wird festgelegt, welche Testszenarien bei den Tests unbedingt durchgeführt werden müssen und welche bei Zeit- oder Budgetmangel – mit bekanntem Risiko – vernachlässigt werden können.

Testfälle mehrfach verwenden
Testfälle sollten bis zum Beginn der Programmierung beschrieben sein und den Entwicklern als konstruktive Qualitätssicherungsmaßnahme zur Verfügung gestellt werden. Auf diese Weise kann das Entwicklerteam in seiner Entwicklungsumgebung eine große Anzahl der Tests mit den Testfällen des Testteams durchführen. Die Entwickler benötigen ihrerseits weniger Zeit für den Entwurf von Testfällen und ersparen bei entsprechender Nachweisführung dem Testteam einige Tests (s. Kap. 9.1 und 9.2).

[6] Zur Durchführung von Reviews s. Kap. 8.1.

4.2 Whitebox-Verfahren

Bei der Ermittlung der Testfälle anhand von Blackbox-Methoden bleiben Programminterna unberücksichtigt. Nicht so bei den Whitebox-Methoden, bei denen Programmstrukturen analysiert werden. Zudem stellen sich im Rahmen der Whitebox-Verfahren interessante Fragen nach der Qualität der durchgeführten Testfälle:

- Gibt es Programmteile, die beim Test nicht durchlaufen wurden?
- Gibt es Programmcode, der nie durchlaufen werden kann?

Testüberdeckungsgrade beantworten diese Fragen. Die Messung eines Testüberdeckungsgrades (Code Coverage Test) stellt fest, welcher Code bzw. wie viel Prozent des Codes eines Programms mit den durchgeführten Tests durchlaufen wurde.

Wenn sich beim Test herausstellt, dass ein Teil des Codes nicht durchlaufen wurde, kann das zwei Ursachen haben. Entweder sind die Testfälle unzureichend, dann müssen zusätzliche definiert werden, oder der Programmcode ist falsch, so dass die betreffenden Statements nie erreicht werden können. In diesem Fall muss der Programmcode angepasst werden.

4.2.1 Einfache Testüberdeckungsgrade

Beim Testen können verschiedene Testüberdeckungsgrade mit unterschiedlicher Aussagekraft bestimmt werden. Am häufigsten werden die Anweisungs-, Zweig- und Bedingungsüberdeckung als Testmetrik eingefordert, weil sie mit akzeptablem Aufwand nachweisbar sind.

Anweisungsüberdeckung (C0, Statement Coverage):
Der Anteil der Anweisungen, die durch eine Menge von Testfällen ausgeführt wurde.

Eine Anweisungsüberdeckung von 100 % bedeutet, dass alle ausführbaren Anweisungen beim Testen ausgeführt wurden.

Entscheidungsüberdeckung (Decision Coverage):
Der Anteil an Entscheidungsausgängen, der durch eine Menge von Testfällen herbeigeführt wurde.

Eine Entscheidungsüberdeckung von 100 % bedeutet, dass jedes mögliche Ergebnis einer Entscheidung (wahr, falsch) mindestens einmal eingetroffen ist.

Zweigüberdeckung (C1, Branch Coverage):
Der Anteil der Zweige, der durch eine Menge von Testfällen ausgeführt wurde.

Tab. 4.10 Pseudocode

Pseudocode
if (STAMMKUNDE = “JA“
or
GESAMTPREIS > 30.000)
then
RABATT = 3%
endif

100 % Zweigüberdeckung entspricht 100 % Entscheidungsüberdeckung; 100 % Zweigüberdeckung umfasst 100 % Anweisungsüberdeckung.

Eine Entscheidung kann auf mehreren atomaren Teilbedingungen basieren, wie z. B. das if-Statement in Tab. 4.10. Diese atomaren Teilbedingungen werden bei den verschiedenen Bedingungsüberdeckungen (Condition Coverage) untersucht.

Einfache Bedingungsüberdeckung (C2):
Der Anteil an atomaren Teilbedingungsausgängen, der durch eine Menge von Testfällen herbeigeführt wurde.

100 % einfache Bedingungsüberdeckung bedeutet, dass jedes atomare Entscheidungskriterium einer Entscheidung mindestens einmal den Wert „wahr“ und „falsch“ hatte.

Die C2-Testüberdeckung ist als alleiniges Qualitätssicherungsmaß nicht befriedigend, weil weder die Zweig- noch die Anweisungsüberdeckung in diesem Maß enthalten sind.

Mehrfachbedingungsüberdeckung (C3):
Der Anteil von Kombinationen der atomaren Teilbedingungen einer Bedingung, der durch eine Menge von Testfällen ausgeführt wurde.

Wenn eine Bedingung n atomare Teilbedingungen enthält, gibt es 2^n Kombinationsmöglichkeiten. Z. B. gibt es bei 3 Teilbedingungen 8 Kombinationen; wenn mit den ausgeführten Testfällen 6 davon getestet werden, ergibt das eine Mehrfachbedingungsüberdeckung von 75 %.

Ein Beispiel für Testüberdeckungsgrade

Ein Beispiel soll die Bedeutung und die Unterschiede der Testüberdeckungsgrade verdeutlichen. Dazu betrachten wir die folgende Anforderung: Einem Kunden wird beim Kauf eines Autos ein Rabatt von 3 % gewährt, wenn für ihn das Stammkundenkennzeichen gesetzt ist oder der Gesamtpreis für die Bestellung über 30.000 € liegt. Die Anforderung ist in Tab. 4.10 in Pseudocode umgesetzt.

Die Forderung der Anweisungsüberdeckung bedeutet, dass jedes ausführbare Statement mindestens einmal durchlaufen wird. Um das nachweisen zu können, legen wir den ersten Testdatensatz fest, der das Stammkundenkennzeichen mit „JA“ und einen Gesamtpreis von 10 € enthält:

Tab. 4.11 Anweisungsüberdeckung

Anweisungsüberdeckung	1.
if (STAMMKUNDE = "JA"	
or	X
GESAMTPREIS > 30.000)	
then	
RABATT = 3%	X
endif	

Testdatensatz	1
STAMMKUNDE	„JA"
GESAMTPREIS	10

Mit diesem Testdatensatz spielen wir Testtool, d. h. wir versehen diejenigen Anweisungen mit einem „X", an denen wir aufgrund der Testdaten vorbeikommen (Tab. 4.11). Somit ist die 100 %-ige C0-Testüberdeckung für unser Beispiel nachgewiesen, weil jedes ausführbare Statement einmal ausgeführt wurde.

Weil der „Ja"-Zweig des Codesegments durchlaufen wird, ist aber nur eine Zweigüberdeckung von 50 % erreicht.

Für den Nachweis der C1-Testüberdeckung von 100 % ergibt sich ein Problem. Wie kann mit unserem Tool nachgewiesen werden, ob der „Nein"-Zweig der Entscheidung jemals durchlaufen wird? Ein „X" im „Nein"-Zweig kann nicht gesetzt werden, weil er keine ausführbare Anweisung enthält. Hier wird deutlich, dass die Testüberdeckung von der Art der Programmierung und dem Testtool abhängt.

Es gibt zwei Lösungen für das Problem: Entweder wird eine Programmiervorgabe erlassen, die „tote" Zweige im Code verbietet, oder es wird ein intelligentes Werkzeug zur Messung der Testüberdeckung eingesetzt, das beim Instrumentieren[GL] des Programms einen solchen Zweig erkennt und ihn automatisch ergänzt.

In unserem Beispiel entscheiden wir uns für die erste Lösung; die Anweisung „CONTINUE"[7], die „nichts tut", wird in den Programmcode eingebaut (Tab. 4.12). Leider ist jetzt mit der CONTINUE-Anweisung unsere 100 %-Anweisungsüberdeckung futsch.

Für eine 100 %-C1- und C0-Überdeckung wird ein zweiter Testdatensatz benötigt:

Testdatensatz	1	2
STAMMKUNDE	„JA"	„NEIN"
GESAMTPREIS	10	10

Mit diesen beiden Testdatensätzen notiert unser Testtool eine Anweisungs- und Zweigüberdeckung von 100 % (Tab. 4.13).

[7] Was „alte" COBOL-Kenner erfreuen wird.

Tab. 4.12 Zweigüberdeckung (1)

Zweigüberdeckung	1.
if (STAMMKUNDE = “JA“	
or	X
GESAMTPREIS > 30.000)	
then	
RABATT = 3%	X
else	
CONTINUE	
endif	

Tab. 4.13 Zweigüberdeckung (2)

Zweigüberdeckung	1.	2.
if (STAMMKUNDE = “JA“		
or	X	X
GESAMTPREIS > 30.000)		
then		
RABATT = 3%	X	
else		
CONTINUE		X
endif		

Soll für unser Miniprogramm die C2-Testüberdeckung nachgewiesen werden, so muss jede atomare Teilbedingung in der if-Abfrage mindestens einmal wahr und einmal falsch sein. Das erreichen die beiden folgenden Testdatensätze, wobei der erste aus den Testdaten für den Nachweis der Zweigüberdeckung stammt:

Testdatensatz	1		3
STAMMKUNDE	„JA“		„NEIN“
GESAMTPREIS	10		44.444

Unser Testtool notiert mit diesen Daten hinter jeder Teilbedingung ein „w“ für wahr oder ein „f“ für falsch (s. Tab. 4.14). Wie in Tab. 4.14 zu erkennen ist, wird der zweite Testdatensatz (STAMMKUNDE = „NEIN“, GESAMTPREIS = 10) zur einfachen Bedingungsüberdeckung nicht benötigt.

Das zeigt, wie schwach dieses Testmaß ist, weil die C1-Testüberdeckung nicht in der C2-Testüberdeckung enthalten ist. Die einfache Bedingungsüberdeckung sollte daher immer in Verbindung mit der Zweigüberdeckung gefordert werden.

Tab. 4.14 Einfache Bedingungsüberdeckung

Einfache Bedingungsüberdeckung	1.	3.
if (STAMMKUNDE = “JA“	w	f
or		
GESAMTPREIS > 30.000)	f	w
then		
RABATT = 3%	X	X
else		
CONTINUE		
endif		

Tab. 4.15 Mehrfache Bedingungsüberdeckung

Mehrfachbedingungsüberdeckung	1.	2.	3.	4.
if (STAMMKUNDE = “JA“	w	f	f	w
or				
GESAMTPREIS > 30.000)	f	f	w	w
then				
RABATT = 3%	X		X	X
else				
CONTINUE		X		
endif				

Wenn der Test für unser Beispielprogramm die Mehrfachbedingungsüberdeckung C3 liefern soll, d. h. alle Kombinationen der Einzelentscheidungen der if-Abfrage getestet werden sollen, wird neben den drei bisherigen ein vierter Testdatensatz benötigt:

Testdatensatz	1	2	3	*4*
STAMMKUNDE	„JA“	„NEIN“	„NEIN“	*„JA“*
GESAMTPREIS	10	10	44.444	*44.444*

Der Testlauf mit diesen vier Testdatensätzen weist den C3-Testüberdeckungsgrad für unser Programm nach, der die C0-, C1- und C2-Überdeckungsgrade einschließt (Tab. 4.15).

4.2.2 Weitere Testüberdeckungsgrade

Bei komplexen Bedingungen kann es vorkommen, dass die elementaren Bedingungen voneinander abhängen und nicht alle Kombinationen für den C3-Testüberdeckungsgrad mit Testdaten erreicht werden können. In diesem Fall werden nur die elementaren Be-

dingungen in Testfällen berücksichtigt, die sich im Ergebnis des Gesamtausdrucks auswirken. Bei einer solchen Reduzierung der möglichen Bedingungskombinationen wird von einer **bestimmenden Bedingungsüberdeckung** gesprochen.

Neben den beschriebenen Testüberdeckungsmaßen gibt es noch weitere, wie zum Beispiel die **Pfadüberdeckung** (C4) (Path Coverage) oder die **Schleifenüberdeckung** (Loop Coverage). Diese sind aber aufgrund des hohen Testaufwands nicht praktikabel und werden hier nicht weiter betrachtet.[8]

Die bisher besprochenen, aus der „alten", nicht objektorientierten Programmierwelt bekannten Testüberdeckungsgrade, begründen sich in den Kontrollflüssen eines Programms (Zweige, Bedingungen, Pfade,…). Auf die objektorientierten Programmiersprachen übertragen, sind diese Metriken nicht immer befriedigend, weil die Besonderheiten der Objektorientierung von ihnen nicht berücksichtigt werden.

Die Testmaße C0 bis C3 können auf den Programmcode jeder Methode einer Klasse angewendet werden. Mit dem Nachweis einer dieser Metriken ist auch jede Methode mindestens einmal ausgeführt worden (**Methodenüberdeckung,** Method Coverage), was im Klassentest (s. Kap. 9.1) als Qualitätssicherungsnachweis gefordert wird.

Schwierig wird ein vollständiger Test von objektorientierten Software-Komponenten durch die Kapselung und Vererbung von Attributen und vielen kleinen[9] Methoden sowie die komplexen Kommunikationsmöglichkeiten unter den verschiedenen Objekten. Objektorientierte Überdeckungsgrade finden in der Praxis (noch?) geringe Anwendung, wie zum Beispiel:

- Jedes Objektattribut wird mindestens einmal modifiziert (Attribute Coverage).
- Jede Nachricht wird mindestens einmal gesendet (Interface Coverage).
- Jeder Parameter jeder Nachricht wird mindestens einmal modifiziert (Parameter Coverage).

4.2.3 Werkzeuge für die Testüberdeckungsgradmessung

Der Nachweis von Testüberdeckungsgraden ist ohne ein **Code Coverage Tool**, das es für jede gängige Programmiersprache gibt, nicht praktikabel. Kommerzielle Tools sind unter [URL: TestToolReview] in der Rubrik „Test Code- und Coverage-Analyse" zu finden. Speziell für Java gibt es Open Source Tools unter [URL: TestToolsJava] in der Rubrik „Code Coverage".

Der werkzeugunterstützte Nachweis einer Testüberdeckung erfolgt in vier Schritten:

[8] Die bestimmende Bedingungsüberdeckung und die Pfadüberdeckung sind zum Beispiel in [Spillner_2012] beschrieben.

[9] Klein im Vergleich mit den Code-Sequenzen prozessorientierter Programmiersprachen.

1. Schritt: Der Sourcecode wird beim Kompilieren vom Code Coverage Tool **instrumentiert**, d. h. er wird so erweitert, dass die ausgeführten Anweisungen, Entscheidungen und/oder Entscheidungskriterien protokolliert werden können.
2. Schritt: Alle nach den Blackbox-Verfahren definierten Testszenarien werden mit den instrumentierten Programmen ausgeführt, wobei die Testdurchläufe vom Code Coverage Tool protokolliert werden. Für die erste Messung des Testüberdeckungsgrades werden zunächst einmal keine zusätzlichen Testfälle benötigt, denn wenn die bis dahin ermittelten Testfälle richtig und vollständig sind, wird mit ihnen die geforderte Testüberdeckung eventuell schon erreicht.
3. Schritt: Sind alle Funktionstests durchgeführt, wird anhand der Testprotokolle überprüft, ob die gewünschte Testüberdeckung erreicht wurde. Nehmen wir der Einfachheit halber C0, d. h. jedes ausführbare Statement sollte durchlaufen worden sein. Wurde eine Anweisung nicht durchlaufen, so kann es dafür drei Erklärungen geben:
   3.a Die Testfälle sind nicht vollständig, denn ein neuer Testfall, der mit seinen Testdaten die fehlende Programmanweisung erreicht, kann gefunden werden.
   3.b Es gibt überflüssigen Programmcode, weil kein Testfall und somit kein Testdatensatz gefunden werden kann, der die bisher nicht ausgeführte Anweisung erreicht.
   3.c Der kritische Fall, dessen Eintreten den Aufwand für die Testüberdeckungsgradmessung rechtfertigt, liegt in der dritten Möglichkeit. Es gibt einen Testfall, der das nicht ausgeführte Statement eigentlich hätte erreichen müssen. Hier liegt ein Programmierfehler vor, der in den vorhergehenden Tests nicht gefunden wurde.
4. Schritt: Anhand der Analyseergebnisse aus dem 3. Schritt werden Testfälle und Testdaten ergänzt und Programmfehler berichtigt. Anschließend werden die Tests wiederholt. Hier ist es von Vorteil, wenn die Tests mit Testrobotern automatisiert sind (s. Kap. 14.3).

4.2.4 Qualitätsanforderungen zu den Whitebox-Verfahren

Die im Testkonzept eines Projektes zu definierenden Testüberdeckungsgrade beschreiben Qualitätsanforderungen an die durchgeführten Tests und stellen zugleich Testendekriterien dar. Zum Beispiel können folgende Qualitätsanforderungen für Programmmodule festgelegt werden, für die allerdings eine Risikoanalyse (s. Kap. 6) Voraussetzung ist:

- Für alle Module der Risikoklasse A ist eine C1-Testüberdeckung von 100 % nachzuweisen.
- Für alle Module der Risikoklasse B und C ist eine C1-Testüberdeckung von 80 % nachzuweisen.
- Sollte die 100 %-Testüberdeckung für ein Modul aus technischen Gründen nicht nachweisbar sein, sind die nicht durchlaufenen Programmteile zu dokumentieren und sowohl vom Projektleiter als auch vom Qualitätssicherungsverantwortlichen zu genehmigen.

Eine Erfahrung zum C1-Testüberdeckungsgrad

Die kompliziert klingende Formulierung zum letzten Punkt der Qualitätsanforderungen beruht auf einer Projekterfahrung.

In einem Projekt war laut vertraglich bindendem Testkonzept eine C1-Testüberdeckung von 100 % nachzuweisen. In jedes Programm mussten aber standardisierte Code-Blöcke zur Fehlerbearbeitung und zur Kommunikation mit dem verwendeten Transaktionsmonitor eingebunden werden. Diese Programmergänzungen bewirkten, dass je nach Umfang des neu programmierten Codes nur eine Testüberdeckung zwischen 60 und 80 % erreicht werden konnte. Der Grund lag darin, dass die Code-Blöcke mit „normalen" Testdaten nicht durchlaufen wurden. Zudem sollten sie im Rahmen der anstehenden fachlichen Tests auch nicht mitgetestet werden, weil sie schon viele Jahre produktiv im Einsatz waren.

Die ursprüngliche Qualitätsanforderung, 100 % Zweigüberdeckung, war somit nicht haltbar. Das Testkonzept musste überarbeitet werden und erhielt den Zusatz: „Sollte aus technischen Gründen die 100 %-Überdeckung für ein Modul nicht nachweisbar sein,…".

Moral von der Geschicht': Qualitätsanforderungen können von Rahmenbedingungen abhängig sein und beide müssen genau überprüft werden.

4.2.5 Empfehlungen zu den Whitebox-Verfahren

Der Nachweis von Testüberdeckungsgraden sollte in frühen Testphasen (z. B. Komponententest und Integrationstest) durchgeführt werden, weil der Sourcecode bereitgestellt werden muss und eine enge Zusammenarbeit mit den Entwicklern erforderlich ist. Zudem sollten keine instrumentierten Programme, wie sie zum Nachweis der Testüberdeckung benötigt werden, in nicht-funktionale Tests gegeben werden, weil die Instrumentierung anwendungsfremden Code erzeugt und u. a. Einfluss auf die Performanz haben kann.

Weil die Anweisungsüberdeckung ein schwaches Testmaß ist und sich Werkzeuge mit der mehrfachen Bedingungsüberdeckung schwer tun, ist der Nachweis der Zweigüberdeckung in Verbindung mit der einfachen Bedingungsüberdeckung eine praktikable Qualitätsanforderung. Manueller Aufwand sollte hierbei vermieden werden, d. h. ein Code Coverage Tool sollte eingesetzt werden.

4.3 Erfahrungsbasierte Testverfahren

Neben den Blackbox- und Whitebox-Methoden zur systematischen Testfallermittlung dürfen die Erfahrungen aus früheren Projekten nicht vergessen werden.

Der erprobte Tester weiß aufgrund seiner Erfahrungen um die Schwachstellen bei der Programmierung und beschreibt Testfälle für Situationen, bei denen mit einer gewissen Wahrscheinlichkeit Fehler gemacht werden (können). Drei Beispiele für intuitive, **erfahrungsbasierte Testfälle**:

Berechnungen
Für unseren Finanzierungsrechner aus dem Kap. 4.1 wäre zum Beispiel interessant zu testen, welches der größtmögliche Kaufpreis ist, der vom Autokonfigurator übergeben wird. Kann der Finanzierungsrechner diese Zahl korrekt darstellen und verarbeiten? Gibt es vielleicht Zahlenüberläufe oder Rundungsfehler?

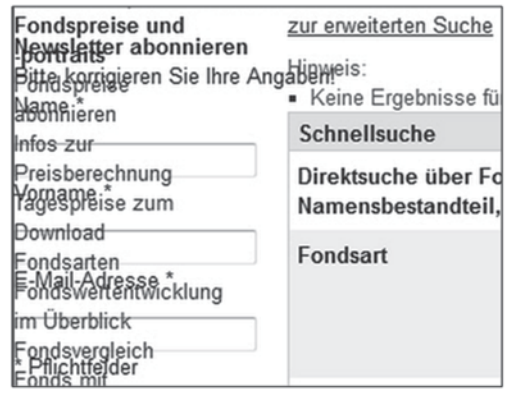
Browser-Navigation
Ein meist erfolgreicher Fehlererwartungstestfall für browser-basierte Anwendungen ist das Aktualisieren des Browser-Fensters während der Bearbeitung einer Transaktion[GL]. Was passiert zum Beispiel, wenn beim Finanzierungsrechner statt der Funktion <Berechnen> die Refresh-Funktion des Browsers ausgeführt wird? Oft passiert es, dass ein Vorgang durch den Aktualisierungs-Button des Browsers unkontrolliert abgebrochen wird.

Zeitzonen
Interessant ist die Reaktion einer App, wenn sie von einem Client aufgerufen wird, der in einer anderen Zeitzone als der Webserver arbeitet. „Verträgt" das System unterschiedliche Zeiteinstellungen?

Damit das Wissen der erfahrenen Tester der Allgemeinheit zur Verfügung steht, sollte von der Qualitätssicherung eine Liste der „beliebten" und bekannten Fehlerquellen erstellt und in Form einer Checkliste fortgeschrieben werden. Eine Checkliste zur Fehlererwartung enthält zum Beispiel folgende Punkte:

✓ Checkliste Fehlererwartung
- Versuche, Divisionen durch Null zu erzeugen.
- Versuche, Zahlenüberläufe zu erzeugen.
- Versuche, Rundungsfehler zu erzeugen.
- Drücke in beliebigen Situationen den Refresh-Button des Browsers.
- Führe in kritischen Situationen die Navigationsfunktionen des Browser aus (Vor- und Zurückblättern).
- Führe mehrmals hintereinander dieselbe Funktion aus, ohne dazwischen andere Aktionen durchzuführen.
- Halte Tasten dauerhaft gedrückt (sowohl bei Eingabefeldern als auch Funktionstasten).
- Gebe in einem Formular HTML-Zeichen ein: > < " &
  (Die Anwendung muss diese Zeichen abweisen. Wenn sie als HTML-Zeichen interpretiert werden, existiert eine Sicherheitslücke, s. Kap. 9.10, Sicherheitstest.)
- Rufe die App zeitzonen- und länderübergreifend auf.
- Drucke über mehrere Seiten.
  (Der Seitenumbruch hat bei mir den ersten Test noch nie bestanden!)
- Schau Dir die Darstellung der Anwendung in verschiedenen Sprachen an.
  (Textlängen verändern sich.)
- Wähle auf unterschiedlichen Endgeräten unterschiedliche Hintergründe (Themes) aus.

Abb. 4.4 überlagerte Felder



(Z. B. schwarze Schrift auf schwarzem Grund kommt immer gut.)

- Verkleinere auf Formularseiten das Browserfenster und schaue, ob sich Felder überlagern, wie in Abb. 4.4.

Anhand einer solchen Checkliste zur Fehlererwartung werden erfahrungsbasierte Testfälle beschrieben, um die mit Blackbox- und Whitebox-Methoden entworfenen Testfälle zu ergänzen.

4.4 Zusammenfassung

- Blackbox- und Whitebox-Verfahren stellen Methoden zur systematischen Testfallerstellung bereit.
- Mit den Blackbox-Verfahren werden Testfälle aus den Anforderungen abgeleitet, ohne dass die Programmstrukturen des Testobjektes untersucht werden.
- Zu den Blackbox-Verfahren gehören die Äquivalenzklassenanalyse, die Grenzwertanalyse, die Ursache-Wirkungs-Analyse sowie die Analyse von Zustandsdiagrammen und Anwendungsfalldiagrammen.
- Mit der Ursache-Wirkungs-Analyse werden Testfälle aus Kombinationen gültiger Äquivalenzklassen abgeleitet und in Entscheidungstabellen dargestellt.
- Komplexe Entscheidungstabellen werden systematisch um unnötige und unmögliche Kombinationen reduziert.
- Die Testfallerstellung mit Hilfe von Blackbox-Methoden ist eine effiziente QS-Maßnahme, denn sie setzt vollständige und detaillierte Anforderungsbeschreibungen voraus und deckt bei der Beschreibung der erwarteten Testergebnisse schnell Mängel in den Spezifikationen auf.
- Mit Whitebox-Verfahren werden Testfälle aus Programmstrukturen abgeleitet.

- Testüberdeckungsgrade geben Auskunft über die Qualität der durchgeführten Tests. Die Methoden-, Anweisungs-, Zweig- und Bedingungsüberdeckung haben am meisten Bedeutung erlangt.
- Der Nachweis der Testüberdeckungsgrade kann nur mit Code Coverage Tools effizient erbracht werden.
- Bei der intuitiven Methode der Fehlererwartung beschreiben erfahrene Tester Testfälle für Situationen, in denen erwartungsgemäß Fehler auftreten. Diese „beliebtesten" Fehler werden in einer regelmäßig zu überarbeitenden Checkliste festgehalten.

Paarweises Testen

5

Flugzeuge fliegen, weil Wirbel stets paarweise auftreten.

Wirbeltheorie – Erkenntnis aus der Aerodynamik

Weil Testressourcen begrenzt sind, können nicht immer alle Kombinationen von Eingabemöglichkeiten oder Systemparametern einer Anwendung mit Testfällen überprüft werden. Daher muss man sich Gedanken über eine sinnvolle Reduzierung machen.

Eine Methode ist das Pairwise-Verfahren, das schon im Abschn. 4.1.3 zur Ursache-Wirkungs-Analyse erwähnt ist. Im dortigen Beispiel geht es darum, dass Bedingungen, die bekannterweise voneinander abhängig sind, kombiniert und in einer Entscheidungstabelle dargestellt werden. Wird diese Entscheidungstabelle aus Sicht der Testdurchführung zu groß, kann sie nach der Pairwise-Methode reduziert werden.

Viel interessanter ist aus meiner Sicht die Betrachtung von unabhängigen Systemparametern. Um mit den beliebten Entwicklervokabeln „eigentlich“ und „sollte“ zu arbeiten: Es gibt in einem Software-System Parameter, die sich untereinander eigentlich nicht beeinflussen sollten.

Zum Beispiel sollte es eigentlich für die Funktionalität eines Webshops egal sein, ob der Käufer online in Amerika, Indien oder Deutschland einkauft, ob er ein Notebook, einen Tablet-PC oder ein Smartphone – mit welchem Betriebssystem auch immer – benutzt und welchen Browser er einsetzt. Gut, der geneigte Tester testet das lieber. Aber wenn er alle möglichen Kombinationen dieser Systemkomponenten testen will, wird er nie zu einem Ende kommen. Wenn er die letzte mögliche Kombination geprüft hat, sind schon fünf neue Mobilgeräte, zwei neue Betriebssystemversionen und drei neue Browserversionen auf dem Markt.

Daher ist es ratsam die Testfälle nach bestimmten Verfahren so zu reduzieren, dass eine hinreichend gute, wohldefinierte Testüberdeckung über die Kombinationen der zu testenden Komponenten erreicht wird. Eine systematische Testfallreduktion kann mit dem

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_5

Pairwise-Verfahren, mit orthogonalen Feldern, dem Klassifikationsbaumverfahren und mit dem N-tupelweisen Testen erreicht werden.

Das Paarweise Testen kommt besonders bei den im Kap. 9.11 beschriebenen Interoperabilitätstests (Cross-Browser-Test, Cross-Device-Test) zum Einsatz.

5.1 Verfahren zum Paarweisen Testen

Beim **Paarweisen Testen** werden die Testfälle so ausgewählt, dass jede Ausprägung eines Parameters mit jeder Ausprägung eines anderen Parameters mindestens einmal kombiniert wird.

Das Paarweise Testen gehört zu den Blackbox-Verfahren, wird aber separat in diesem Kap. 5 behandelt, weil wir mit dem Paarweisen Testen ein ganz spezielles Ziel verfolgen:

> ► Stelle mit vertretbarem Aufwand und einer hohen Wahrscheinlichkeit sicher, dass Kombinationen von voneinander unabhängigen Systemparametern keinen Einfluss auf das Testobjekt haben.

Für eine Web-App zum Beispiel bedeutet Paarweises Testen, dass nicht alle möglichen Kombinationen von z. B. Betriebssystem, Browser, Spracheinstellungen,… getestet werden. Durch systematische Paarbildung wird die Wahrscheinlichkeit des Eintretens eines Fehlers aufgrund zweier inkompatibler Parameter drastisch reduziert, immer im Hinterkopf, dass die Parameter „eigentlich“ voneinander unabhängig sein „sollten“.

Bleibt die Frage, wie diese Paare systematisch und vollständig gefunden werden können.

5.1.1 Pairwise-Verfahren

Das **Pairwise-Verfahren** liefert Testfälle, so dass sichergestellt ist, dass jede Ausprägung eines Parameters mit jeder Ausprägung eines anderen Parameters mindestens einmal kombiniert wird. Das folgende Beispiel erklärt die Methode. Gegeben sind drei Parameter Par1, Par2 und Par3 mit den Ausprägungen:

- Par1: {1, 2, 3, 4}
- Par2: {a, b, c}
- Par3: {i, ii}

Wenn alle Kombinationen getestet werden sollen, ergeben sich 24 Testfälle (Tab. 5.1). Die Anzahl der Testfälle soll nun so reduziert werden, dass jede Ausprägung eines Parameters mit jeder Ausprägung eines anderen Parameters mindestens einmal in Kombination auftritt.

Tab. 5.1 Alle Parameter-Kombinationen

TF	Par1	Par2	Par3
1.	1	a	i
2.	1	a	ii
3.	1	b	i
4.	1	b	ii
5.	1	c	i
6.	1	c	ii
7.	2	a	i
8.	2	a	ii
9.	2	b	i
10.	2	b	ii
11.	2	c	i
12.	2	c	ii
13.	3	a	i
14.	3	a	ii
15.	3	b	i
16.	3	b	ii
17.	3	c	i
18.	3	c	ii
19.	4	a	i
20.	4	a	ii
21.	4	b	i
22.	4	b	ii
23.	4	c	i
24.	4	c	ii

In unserem Beispiel werden von oben nach unten die Zeilen gestrichen, bei denen alle Pärchen der Parameterkombination in einer anderen Zeile vorkommen. Das Vorgehen wird in Tab. 5.2 durchexerziert, wobei in der ersten Zeile mit der Prüfung und dem Streichen begonnen wird. Als Ergebnis erhalten wir 12 Testfälle.

Das Pairwise-Verfahren kann mit unterschiedlichen Algorithmen durchgeführt werden; je nachdem, welcher angewendet wird, kann das konkrete Ergebnis ein anderes sein. Zum Beispiel werden andere Zeilen in der Tabelle gestrichen, wenn

- in der Tabelle nicht die erste Zeile gestrichen wird, sondern die erste Zeile prinzipiell stehen bleibt und in der zweiten mit der Prüfung begonnen wird, oder
- wenn nicht von oben nach unten, sondern von unten nach oben geprüft und gestrichen wird, oder
- wenn die Kombinationen in der Tabelle anders gebildet werden, d. h. die Spalten in einer anderen Reihenfolge stehen.

Tab. 5.2 Streichen von Parameter-Kombinationen

TF	Par1	Par2	Par3	Prüfungsschritte top-down	Ergebnis
1.	1	a	i	1-a kommt in Zeile 2 vor, 1-i kommt in Zeile 3 vor, a-i kommt in Zeile 2 vor	Streichen
2.	1	a	ii	1-a kommt nicht mehr vor (wurde mit 1. gestrichen) und die beiden anderen Pärchen müssen somit nicht mehr geprüft werden	Behalten
3.	1	b	i	1-b kommt in Zeile 4 vor, 1-i kommt in Zeile 5 vor, b-i kommt in Zeile 9 vor	Streichen
4.	1	b	ii	1-b kommt nicht mehr vor	Behalten
5.	1	c	i	1-c kommt nicht mehr vor	Behalten
6.	1	c	ii	1-c kommt in Zeile 5 vor, 1-ii kommt in Zeile 2 vor, c-ii kommt in Zeile 12 vor	Streichen
7.	2	a	i	…	Streichen
8.	2	a	ii	–	Behalten
9.	2	b	i	–	Streichen
10.	2	b	ii	–	Behalten
11.	2	c	i	–	Behalten
12.	2	c	ii	–	Streichen
13.	3	a	i	–	Streichen
14.	3	a	ii	–	Behalten
15.	3	b	i	–	Streichen
16.	3	b	ii	–	Behalten
17.	3	c	i	–	Behalten
18.	3	c	ii	–	Streichen
19.	4	a	i	–	Behalten
20.	4	a	ii	–	Streichen
21.	4	b	i	–	Behalten
22.	4	b	ii	–	Streichen
23.	4	c	i	–	Streichen
24.	4	c	ii	–	Behalten

Beispiel: Pairwise-Verfahren für den Autokonfigurator

Soweit das Verfahren im Allgemeinen. Nehmen wir nun für unseren Autokonfigurator an, dass er nur im Extranet bei den Autohäusern unseres Unternehmens eingesetzt wird. Den Autohäusern ist die IT-Infrastruktur vorgeschrieben, es werden folgende Systemkomponenten unterstützt:

Tab. 5.3 Reduzierte Testfälle Autokonfigurator

TF	Browser	BS	Sprache
1.	Firefox 17	Windows XP	Deutsch
2.	Firefox 17	Windows 7	Deutsch
3.	Firefox 17	Windows 8	Englisch
4.	Firefox 18	Windows XP	Deutsch
5.	Firefox 18	Windows 7	Deutsch
6.	Firefox 18	Windows 8	Englisch
7.	IE 9	Windows XP	Deutsch
8.	IE 9	Windows 7	Deutsch
9.	IE 9	Windows 8	Englisch
10.	IE 10	Windows XP	Englisch
11.	IE 10	Windows 7	Englisch
12.	IE 10	Windows 8	Deutsch

- Betriebssystem: Windows XP, Windows 7, Windows 8
- Browser in Version: Firefox 17, Firefox 18, IE 9, IE10
- Sprache (gilt immer gleichzeitig für Betriebssystem, Browser und Autokonfigurator): Englisch, Deutsch

Wenn wir damit die drei in Tab. 5.2 aufgeführten abstrakten Parameter ersetzen, ergeben sich die 12 konkreten Testfälle in Tab. 5.3. Unseren Autokonfigurator würden wir also mit den wichtigsten Anwendungsfällen in den 12 gefundene Systemkonstellationen testen. Das Risiko besteht jetzt noch darin, dass wir Kompatibilitätsfehler nicht finden, die beim Zusammenspiel von drei Komponenten auftreten, die wir zufällig nicht mit unseren Testfällen abdecken. Im Beispiel wird u. a. das Tripel IE9/Windows 7/Englisch nicht getestet.

5.1.2 Orthogonale Felder

Mit orthogonalen Feldern können recht einfach Paare von Parametern in Testfällen gebildet werden, so dass sichergestellt ist, dass jeder mit jedem… aber diese Bedingung kennen wir ja schon.

Was ist ein orthogonales Feld?
Ein **orthogonales Feld** ist eine zweidimensionale Matrix, mit der Eigenschaft, dass jeweils zwei Spalten senkrecht aufeinander stehen, d. h. ihr Skalarprodukt 0 ist.

Das kennen wir aus der Geometrie. Zwei Vektoren sind zueinander orthogonal, wenn der Winkel zwischen ihnen 90° beträgt, d. h. ihr Skalarprodukt 0 ist. Egal, genug Mathematik. Für das Testen bedeutet das: In einem orthogonalen Feld entsprechen die Zeilen den Testfällen und die Spalten den Parametern, die kombiniert werden sollen. Die Anzahl

Tab. 5.4 Orthogonales Feld $L_4(2^3)$

TF	Par1	Par2	Par3
1.	1	1	1
2.	1	2	2
3.	2	1	2
4.	2	2	1

Tab. 5.5 Orthogonales Feld $L_{16}(4^3)$

1	1	1
2	2	1
3	3	1
4	4	1
1	2	2
2	1	2
3	4	2
4	3	2
1	3	3
2	4	3
3	1	3
4	2	3
1	4	4
2	3	4
3	2	4
4	1	4

der Ausprägungen, die ein Parameter annehmen kann, ist eine natürliche Zahl n, der sogenannte Level. In jeder Spalte kommt jede Ausprägung des Parameters gleich oft vor, und jede Ausprägung der einen Spalte bildet genau einmal ein Paar mit jeder Ausprägung der anderen Spalte.

Tabelle 5.4 zeigt das Beispiel $L_4(2^3)$, ein einfaches orthogonales Feld. Sie zeigt, dass man 4 Testfälle benötigt, um 3 Parameter mit jeweils 2 Ausprägungen (Level n = 2) zu testen.

Wie kommt man an orthogonale Felder?
Orthogonale Felder kann man mit viel Konzentration selbst erstellen, aus Lateinischen Quadraten[GL] „basteln“ oder man kann sich im Internet halbwegs passende zu seinen Testfällen suchen, z. B. bei [URL: OrthoArray-1] oder [URL: OrthoArray-2]. Bei Letzterem finden wir das orthogonale Feld $L_{16}(4^3)$ (Tab. 5.5). Passt es zu unserem obigen Beispiel?

Wie benutzt man orthogonale Felder beim Testen?
Weil der Parameter „Browser“ vier Ausprägungen hat, brauchen wir ein orthogonales Feld mit dem Level 4. Mit drei Parametern und den genannten Ausprägungen müssten eigentlich 4*3 = 12 Testfälle für den paarweisen Test genügen. Aber wir finden kein orthogona-

Tab. 5.6 Paarweise Testfälle mit orthogonalem Feld

TF	Browser	BS	Sprache
1.	Firefox 17	Windows XP	Englisch
2.	Firefox 18	Windows 7	Englisch
3.	IE 9	Windows 8	Englisch
4.	IE 10	4	Englisch
5.	Firefox 17	Windows 7	Deutsch
6.	Firefox 18	Windows XP	Deutsch
7.	IE 9	4	Deutsch
8.	IE 10	Windows 8	Deutsch
9.	Firefox 17	Windows 8	3
10.	Firefox 18	4	3
11.	IE 9	Windows XP	3
12.	IE 10	Windows 7	3
13.	Firefox 17	4	4
14.	Firefox 18	Windows 8	4
15.	IE 9	Windows 7	4
16.	IE 10	Windows XP	4

les Feld $L_{12}(4^3)$ – gibt es wohl nicht, das müsste dann auch $L_{12}(4^1 3^1 2^1)$ heißen. Also nehmen wir $L_{16}(4^3)$ aus Tab. 5.5, setzen für die Zahlen die Ausprägungen unserer Parameter ein und erhalten die Testfälle in Tab. 5.6.

Die Tab. 5.6 ist nach dem Ersetzen der Parameterausprägungen nicht komplett ausgefüllt, weil wir kein genau passendes orthogonales Feld gefunden haben. Aber TF 10 könnte gestrichen werden, TF 13 ebenfalls. TF 4 könnte mit TF 12 zusammengelegt werden und TF 7 mit TF 15; dann kämen wir auch auf 12 Testfälle wie beim Pairwise-Verfahren.

Man kann das aber auch alles so lassen und die unbesetzten Parameter beliebig auffüllen – wer weiß, vielleicht müssen demnächst ja doch noch ein viertes Betriebssystem und ein oder zwei neue Sprachen getestet werden.

Für Sodoku-Fans: Wer Spaß am paarweisen Testen hat, kann dieses Thema noch vertiefen und sich in die lateinischen Quadrate und orthogonalen Felder einarbeiten, zum Beispiel in [Dustin_2001] und [URL: OATS].

5.1.3 Klassifikationsbäume

Ein weiteres Verfahren, um die Anzahl der Testfälle im Zaum zu halten, ist das **Klassifikationsbaumverfahren**. Dabei werden wie bei dem Paarweisen Testen mehrere Bedingungen gemeinsam getestet.

Zuerst werden die Testbedingungen in Kategorien eingeordnet. Diese Kategorien werden dann auf einzelne Bedingungen (Äquivalenzklassen) heruntergebrochen, die je nach

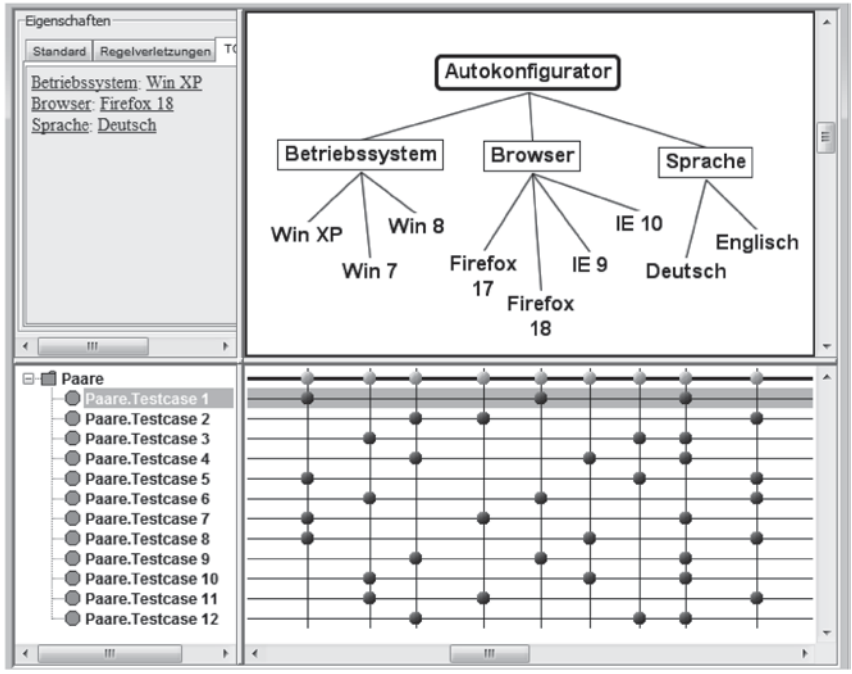


Abb. 5.1 Klassifikationsbaum mit CTE XL

Fachlichkeit selbst wieder heruntergebrochen werden können (verfeinerte Äquivalenzklassen), und so weiter.

Auf diese Art erhält man einen Baum von Klassifikationen, dessen Blätter in konkreten Testfällen so kombiniert werden, dass jede Ausprägung einer Bedingung mindestens einmal getestet wird oder dass jede Ausprägung einer Bedingung mit jeder Ausprägung einer anderen Bedingung als Paar mindestens einmal getestet wird.

Da ein Klassifikationsbaum eine grafische Darstellung besitzt, ist darin schön zu erkennen, welche Klassifikation (oder Paare oder Tripel von Klassifikationen) einem Testfall zugeordnet ist – oder eben noch nicht. Aber je mehr Kategorien und Bedingungen in einem Baum dargestellt werden müssen, desto unübersichtlicher wird das Ganze natürlich. Ohne Werkzeug ist dieses Verfahren schwer handhabbar.

Ein Beispiel für die Klassifikationsbaummethode zu unserem Autokonfigurator ist in Abb. 5.1 mit dem Tool **CTE XL**[1] [URL: CTE XL] dargestellt.

[1] Erstellt mit CTE XL. Ab Sommer 2014 wird das Tool **Testona** heißen.

5.1.4 N-tupelweises Testen

Ein Risiko beim Paarweisen Testen besteht, wenn z. B. ein Fehler in einer Dreierkombination, einer Viererkombination oder einer noch höherwertigeren Kombination von Parametern auftritt, die mehr oder weniger zufällig nicht getestet wird.

Sollen diese Risiken vermieden werden, können die oben beschriebenen Verfahren auf Tripel, Quadrupel oder N-Tupel erweitert werden. Dann handelt es sich aber per Definition nicht mehr um Paarweises Testen, sondern um **N-tupelweises Testen**. Und ohne Werkzeugeinsatz ist die Testfallerstellung dann kein Spaß. Zum Werkzeugeinsatz siehe Kap. 5.3.

5.2 Empfehlungen zum Einsatz des Paarweisen Testens

Wenn nicht gerade sicherheitskritische Systeme getestet werden, sind die Verfahren des Paarweisen Testens m. E. hinreichend. Das N-tupelweise Testen wird im Web- und Mobile-Bereich eher nicht angewendet.

5.2.1 Paarweises Testen im Komponententest

Das Pairwise-Verfahren kann im funktionalen Komponententest (s. Kap. 9.2) angewendet werden, wenn zum Beispiel in einem Formular einer Webseite jede gültige Äquivalenzklasse jedes Feldes mit jeder gültigen Äquivalenzklasse jeden anderen Feldes in einem Testfall ausgeführt werden soll. Aber wer macht das schon, wenn diese Felder aufgrund fachlicher Anforderungen nicht voneinander abhängen?

Wenn es Abhängigkeiten gibt, wird in jedem Fall eine Entscheidungstabelle mit allen möglichen Kombinationen entwickelt und nur wenn diese zu groß wird, kann sie z. B. mit dem Pairwise-Verfahren reduziert werden. Aber Achtung: dabei dürfen keine Wirkungen abhandenkommen!

5.2.2 Paarweises Testen aus technischer Sicht

Einen sehr großen Nutzen hat das Paarweise Testen beim Nachweis der Interoperabilität (s. Kap. 9.11) der verschiedenen Bestandteile eines Software-Systems wie Hardware-Komponenten, Software-Komponenten, Datenbanken, Browser, Betriebssysteme, Sprachversionen.

Besonders beim Mobile-App-Testing sind die beschriebenen Verfahren zur Testfallreduzierung sehr wichtig, ist die Anzahl der zu betrachtenden Parameter mit Browsern, Betriebssystemen, Endgeräteherstellern, Sim-Karten, Verbindungsstärken,… oft sehr, sehr groß.

Tab. 5.7 Fachliche Parameter des Webshops

Auftragsannahme	Bezahlsystem	Lieferant	Logistik	Retoure
Online	Bankeinzug	Lieferant A	DHL	Ja
Telefon	Pay-Pal	Lieferant B	Hermes	Nein
	Visa	Lieferant C	UPS	
	Mastercard	Lieferant D		
	American Express	Lieferant E		
		Lieferant F		

5.2.3 Paarweises Testen aus fachlicher Sicht

Das Paarweise Testen ist nicht nur bei technischen Artefakten einsetzbar, sondern auch aus fachlicher Sicht im Systemtest oder Abnahmetest, insbesondere wenn eine Anwendung über mehrere Systeme hinweg getestet werden muss.

Beispiel: Paarweises Testen im Webshop

Wir haben neben dem Autokonfigurator neuerdings einen Webshop für Autozubehör. Der gesamte Auftragsabwicklungsprozess besteht aus mehreren Schritten, die viele Einzelausprägungen haben können, siehe Tab. 5.7. (Im richtigen Leben sind es natürlich noch viele mehr.)

Im Webshop kann Autozubehör bestellt werden. Es wird mit mehreren Lieferanten zusammengearbeitet. Eine Bestellung kann online oder per Telefon erfolgen. Das System prüft die Verfügbarkeit und leitet die Bestellung an den günstigsten Lieferanten weiter, der die Ware sofort liefern kann. Der Lieferant muss dem Kunden eine Auftragsbestätigung zusenden und die Ware ausliefern und den Bezahlvorgang abwickeln. Der Kunde kann per Bankeinzug, Pay-Pal, Kreditkarte (Visa, Mastercard, American Express) bezahlen. Die Auslieferung der Ware kann mit DHL, Hermes oder UPS erfolgen, das entscheidet der Lieferant. Der Auslieferungsstatus kann im Webshop verfolgt werden. Der Kunde kann die Ware natürlich zurückgehen lassen. Die Retouren-Meldung des Kunden erfolgt über den Webshop, die Abwicklung muss der Lieferant übernehmen. Wenn ein Bestellprozess komplett abgeschlossen ist, berechnet das Webshop-System die Provision und stellt sie monatlich dem Lieferanten in Rechnung.

Die einzelnen Systemkomponenten tauschen ihre Daten über technische Schnittstellen aus, die müssen natürlich im Integrationstest getestet sein.

Dem Modul <Bezahlsystem> ist „eigentlich“ total egal, ob der Auftrag online oder per Telefon reingekommen ist, und erst recht, wer die Bestellung zum Kunden fährt. Die unterschiedlichen Kombinationen „sollten“ keine Auswirkungen auf einen Teilprozess oder gar den Gesamtprozess haben.

Um ganz sicherzugehen, dass der Prozess der Bestellabwicklung immer funktioniert, könnten alle Kombinationen getestet werden; das wären 2*5*6*3*2 = 360 Testfälle. Das ist nicht bezahlbar; daher ist hier aus fachlicher Sicht Paarweises Testen angeraten.

5.3 Werkzeuge zum Paarweisen Testen

Hausaufgabe: Schreiben Sie die 360 errechneten Kombinationstestfälle zum Autokonfigurator-Beispiel auf und wenden Sie das Pairwise-Verfahren manuell an. Das übt, viel Spaß.

Effizienter ist ein Tool, wie zum Beispiel das webbasierte Tool von **Hexawise** [URL: Hexawise]. Die 360 Testfälle aus dem obigen Beispiel sind mit Hexawise auf 30 reduziert worden (s. Tab. 5.8). Die kursiv geschriebenen Werte können beliebig geändert werden, sie haben keinen Einfluss auf die Testüberdeckung von 100 %.

In Hexawise können besondere Abhängigkeiten der Parameter hinterlegt werden, die Einfluss auf die Bildung der Testfälle haben. Das können Parameterkombinationen sein, die immer gemeinsam getestet werden müssen oder nie zusammen auftreten können.

Wenn in unserem Beispiel die Lieferanten D, E und F nicht mit American Express zusammenarbeiten würden, können diese Restriktionen in Hexawise hinterlegt werden. Dadurch würden sich weniger Testfälle ergeben.[2]

Auch N-tupelweises Testen ist mit Hexawise möglich und es können sogar unterschiedliche Verfahren für einzelne Parameter kombiniert werden.

Ein einfaches, skriptbasiertes Open Source Tool unter GPL 2.0, dass sich auf die Pairwise-Methode zur Reduzierung von Entscheidungstabellen konzentriert, ist **ALLPAIRS** von James Bach [URL: ALLPAIRS].

In Abb. 5.1 ist das Beispiel des Autokonfigurators mit dem Klassifikationsbaum-Editor **CTE XL** dargestellt [URL: CTE XL], auch hier erhalten wir 12 Testfälle. In CTE XL können Abhängigkeiten zwischen Parametern festgelegt werden, und neben der paarweisen Testfallgenerierung können Testfälle auch mit anderen Algorithmen auf Basis des zugehörenden Klassifikationsbaums erzeugt werden.

Ein freies, webbasiertes Tool ist auf der Combinatorial Testing Page zu finden [URL: CombTest]. Es ist einfach zu bedienen, allerdings können die Eingaben nicht für die nächste Session gespeichert werden. Zum Ausprobieren des Paarweisen Testens und einiger anderer Algorithmen ist **CombTest** recht spannend.

Eine umfangreiche Liste der Tools, die unter anderem das Paarweise Testen unterstützen, ist unter [URL: Pairwise] zu finden.

[2] Für die Testfälle in Tab. 5.8 wurden diese Restriktionen nicht festgelegt!

Tab. 5.8 Paarweise Testfälle Hexawise

	Auftragsannahme	Bezahlsystem	Lieferant	Auslieferer	Retoure
1	Online	Bankeinzug	Lieferant A	DHL	Ja
2	Telefon	Pay-Pal	Lieferant A	Hermes	Nein
3	Telefon	Visa	Lieferant A	UPS	Ja
4	Online	Mastercard	Lieferant A	DHL	Nein
5	Telefon	American Express	Lieferant A	DHL	Ja
6	Online	Bankeinzug	Lieferant B	Hermes	Ja
7	Online	Pay-Pal	Lieferant B	DHL	Ja
8	Online	Visa	Lieferant B	DHL	Nein
9	Telefon	Mastercard	Lieferant B	UPS	Ja
10	Online	American Express	Lieferant B	Hermes	Nein
11	Online	Bankeinzug	Lieferant C	UPS	Nein
12	Telefon	Pay-Pal	Lieferant C	DHL	Ja
13	*Telefon*	Visa	Lieferant C	Hermes	*Ja*
14	*Telefon*	Mastercard	Lieferant C	Hermes	*Nein*
15	*Online*	American Express	Lieferant C	UPS	*Ja*
16	Telefon	Bankeinzug	Lieferant D	DHL	Ja
17	Online	Pay-Pal	Lieferant D	UPS	Nein
18	*Telefon*	Visa	Lieferant D	Hermes	*Ja*
19	*Online*	Mastercard	Lieferant D	*DHL*	*Nein*
20	*Online*	American Express	Lieferant D	*DHL*	*Ja*
21	Online	Bankeinzug	Lieferant E	DHL	Ja
22	Telefon	Pay-Pal	Lieferant E	Hermes	Nein
23	*Online*	Visa	Lieferant E	UPS	*Ja*
24	*Online*	Mastercard	Lieferant E	*DHL*	*Nein*
25	*Online*	American Express	Lieferant E	*Hermes*	*Nein*
26	Online	Bankeinzug	Lieferant F	DHL	Ja
27	Telefon	Pay-Pal	Lieferant F	Hermes	Nein
28	*Online*	Visa	Lieferant F	UPS	*Nein*
29	*Online*	Mastercard	Lieferant F	*DHL*	*Nein*
30	*Online*	American Express	Lieferant F	*Hermes*	*Ja*

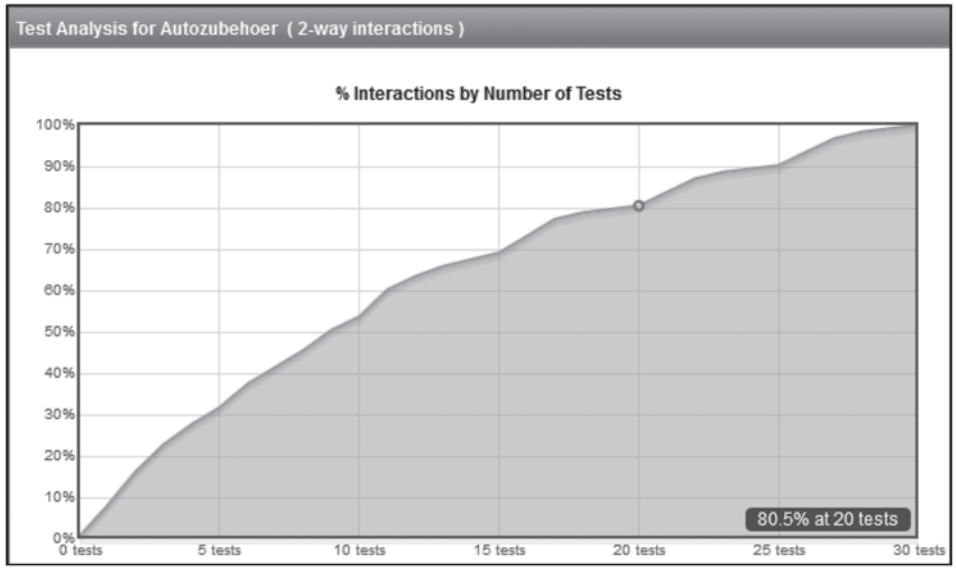


Abb. 5.2 Testüberdeckung Hexawise

5.4 Qualitätsanforderungen zum Paarweisen Testen

Ein Qualitätsmaß für das Paarweise Testen ist die Überdeckung der zu testenden Parameterpaare.

Für unser Beispiel des Webshops mit 30 Testfällen aus Tab. 5.8 liefert Hexawise eine Übersicht für den Nachweis der Testüberdeckung. In Abb. 5.2 kann man sehen, dass mit 20 Testfällen eine Testüberdeckung von 80,5 % der paarweisen Testfallüberdeckung erreicht wird.

5.5 Zusammenfassung

- Mit den Verfahren des Paarweisen Testens wird eine hohe Anzahl von Parameter-Kombinationen systematisch auf ein erträgliches Maß reduziert.
- Die Verfahren des Paarweisen Testens liefern Testfälle, so dass sichergestellt ist, dass jede Ausprägung eines Parameters mit jeder Ausprägung eines anderen Parameters mindestens einmal kombiniert wird.
- Mit dem Pairwise-Verfahren und mit orthogonalen Feldern werden Testfälle mit paarweisen Parameterkombinationen erzeugt.
- Die Klassifikationsbaummethode stellt die Kombinationen von Testfällen grafisch dar.
- Beim N-tupelweisen Testen werden in den Testfällen nicht nur Paare von Parametern kombiniert, sondern auch Tripel, Quadrupel, Quintupel…
- Tools unterstützen das Paarweise Testen und das N-tupelweise Testen.

Risikoanalyse

6

Manchmal muss man einfach ein Risiko eingehen
- und seine Fehler unterwegs korrigieren.

Lee Iacocca (*1924)

Die Vielfalt der QS-Aufgaben und der Aufwand der durchzuführenden QS-Maßnahmen werden in termingedrängten Projekten oft unterschätzt. Folge davon sind voreilige und unkontrollierte Freigaben von Software, Hauptsache Termine halten. Bisher sind aber alle mir bekannten, vom Management verordneten „Risikofreigaben" schief gelaufen, d. h. der vermeintlich eingesparte Testaufwand kam stets in Form von Produktionsfehlern und -ausfällen als Boomerang zurück.

Risikoanalysen sorgen dafür, dass die Freigabe einer Anwendung kein unwägbares Risiko darstellt, sondern die Risiken bei einer Freigabe bekannt und kalkulierbar sind.

6.1 Ziele der Risikoanalyse

Wenn man sich die technische Komplexität einer Webapplikation und die sich daraus ergebenden Testvarianten vor Augen führt, wird deutlich, wie viel Fehler- und somit Risikopotential darin stecken. Um das Risiko einer Software-Freigabe einschätzen und die Balance zwischen Testaufwand und Kosten in einem Projekt halten zu können, müssen Risikoanalysen Bestandteil der Projektplanung sein und zielgerichtet durchgeführt werden.

Mit der Risikoanalyse werden Geschäftsprozesse, Testobjekte, Testfälle und Fehler priorisiert, Risiken transparent gemacht und geeignete QS-Maßnahmen abgeleitet. Zudem stellt die Risikoanalyse sicher, dass die wichtigsten Tests zuerst durchgeführt werden, falls der Testaufwand aus Zeit- oder Budgetgründen reduziert werden muss.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_6

6.2 Grundlagen der Risikoanalyse

Ein Risiko ist die Möglichkeit des Eintretens eines Ereignisses mit negativen Auswirkungen. In der Mathematik wird das Risiko definiert als Eintrittswahrscheinlichkeit des negativen Ereignisses multipliziert mit der potentiellen Schadenshöhe:

$$Risiko = Eintrittswahrscheinlichkeit * Ausma\beta$$

Die Fehlermöglichkeits- und Einflussanalyse (FMEA[1] – Failure Mode and Effects Analysis) misst ein Risiko durch die *Risikoprioritätszahl* (RPZ), indem sie das Risiko mit der Wahrscheinlichkeit der Nichtentdeckung bewertet:

$$RPZ = Eintrittswahrscheinlichkeit * Ausma\beta * Wahrscheinlichkeit\ der\ Nichtentdeckung$$

Jeder der drei Faktoren eines Risikos erhält in der FMEA eine Bewertung zwischen 1 und 10, so dass jedes Risiko eine RPZ zwischen 1 und 1000 besitzt. In Abhängigkeit der Höhe der RPZ werden Maßnahmen festgelegt, mit denen dem Risiko begegnet wird.

6.3 Risikoanalyse in der Software-Entwicklung

Projekt- und Produktrisiken
In einem Software-Entwicklungsprojekt müssen Projektrisiken (zum Beispiel die Beistellungen des Auftraggebers oder die Verfügbarkeit von Ressourcen) und Produktrisiken (zum Beispiel die Fehler in Programmen) bewertet werden.

Die Bestimmung der Risikoprioritätszahl ist für Software-Produkte nicht einfach, denn die Eintrittswahrscheinlichkeit und die Wahrscheinlichkeit der Nichtentdeckung eines nicht bekannten Fehlers sind – wenn überhaupt – sehr schwer zu bestimmen.

ABER: Durch systematische und ausführliche Tests kann die Eintrittswahrscheinlichkeit eines negativen Ereignisses, was in unserem Kontext das Eintreten einer Fehlerwirkung bedeutet, verringert werden. Um im Rahmen der Testplanung die notwendigen Qualitätssicherungsmaßnahmen ergreifen und richtigen Entscheidungen treffen zu können, muss das Ausmaß einer möglicherweise eintretenden Fehlerwirkung abgeschätzt werden. Dazu hat sich in IT-Projekten die ABC-Analyse bewährt, die eine dreistufige Bewertungsskala statt einer zehnstufigen benutzt.

ABC-Risikoanalyse
Durch eine ABC-Risikoanalyse werden die zu bewertenden Objekte in die Risikoklassen (Risikostufen) A, B oder C eingeteilt. Diesen drei Risikoklassen werden Maßnahmen zu-

[1] Die FMEA wird in der DIN 60812 beschrieben ([DIN 60812]).

geordnet, die dem Risiko entgegenwirken sollen. Die zu bewertenden Objekte und zugeordneten Maßnahmen können unterschiedlicher Natur sein, wie folgende Beispiele zeigen:

- Testobjekte werden einer Risikoklasse zugeordnet, um angemessene Qualitätssicherungsmaßnahmen und Testmetriken festlegen zu können. Kritische Komponenten mit hoher Risikoklasse unterliegen strengeren Maß(nahm)en als unkritische.
- Testfälle werden bewertet, um festzulegen, mit welcher Priorität sie zu testen sind. Die wichtigsten Testfälle, d. h. Testfälle, die Fehlerwirkungen mit der höchsten potentiellen Schadenshöhe aufdecken können, müssen zuerst durchgeführt werden.
- Mängel und aufgedeckte Fehler werden priorisiert, um Freigabeentscheidungen treffen zu können. Unkritische Mängel sollten nie den Projektfortschritt oder eine Freigabestufe verhindern.

Beispiele für Risikoklassen

Die beiden folgenden Beispiele beschreiben Definitionen von Risikoklassen, wie sie im Testkonzept eines Projektes festgelegt werden können.

Beispiel 1: Risikoklassen für Mängel in Dokumenten

Wenn in einem Review zur Abnahme einer Anforderungsspezifikation Mängel festgestellt werden, so müssen diese bewertet werden, um die weiteren Schritte im Projekt festlegen zu können. In Tab. 6.1 sind Merkmale festgelegt, mit denen ein in einem Dokument gefundener Fehler einer Risikoklasse zugeordnet wird.

Für jede Risikoklasse werden Maßnahmen und Konsequenzen festgelegt:

- Mängel mit der Risikoklasse A und B verhindern die Freigabe eines Dokumentes und der darauf basierenden Arbeitsergebnisse. Mängel mit der Risikoklasse A und B müssen vor der Freigabe behoben werden.
- Wird bei einem Review ein Fehler der Klasse A festgestellt, so muss nach Überarbeitung des Dokumentes ein erneutes Review durch Experten stattfinden. Für B-Fehler ist die Prüfung durch den Qualitätssicherungsverantwortlichen hinreichend.
- Mängel der Risikoklasse C können bis zu einem im Prüfprotokoll festgelegten Termin beseitigt werden, der nach dem Freigabetermin liegen kann. Die Nachprüfung erfolgt durch den Qualitätssicherungsverantwortlichen.

Tab. 6.1 Risikoklassen für Dokumente

Risikoklasse	Risikomerkmal
A – Hoch	Falsche oder unvollständige Beschreibung
	Fehlende Funktionalität oder fehlender Algorithmus
B – Mittel	Formulierungen, die falsch interpretiert werden können
C – Gering	Schreib- und Formfehler
	Fehlende Erläuterungen

Tab. 6.2 Risikoklassen für Programme

Risikoklasse	Risikomerkmal/Fehlerwirkung
A – Hoch	Geldwerte Nachteile entstehen
	Schadensansprüche können gestellt werden
	Aufträge gehen verloren
B – Mittel	Kunden werden verärgert
	Interessenten werden nicht gewonnen
	Relevante Informationen werden nicht geliefert
C – Gering	Mängel und Schönheitsfehler, die nicht geschäftsrelevant sind

Die in Tab. 6.1 beschriebenen Risikoklassen und damit verbundenen QS-Maßnahmen werden im Dokumententest angewendet (s. Kap. 8).

Beispiel 2: Risikoklassen für Programme
Ein fehlerhaft arbeitendes Programm stellt ein Risiko dar. In Tab. 6.2 sind Risikoklassen festgelegt, die durch das Ausmaß einer potentiellen Fehlerwirkung bestimmt werden. Im Rahmen der Risikoanalyse muss jedes Programm einer Risikoklasse zugeordnet werden, bevor es von der Qualitätssicherung geprüft wird.

Bewerten wir drei Beispielprogramme anhand dieser Risikoklassen:

1. Der Finanzierungsrechner, der durch einen Fehler falsche Finanzierungsraten berechnet, ist in die Risikoklasse A einzuordnen, weil ein finanzieller Schaden entstehen kann.
2. Das Programm, das automatisch Informationsmaterial per E-Mail versendet, wird mit der Klasse B bewertet. Denn wenn ein Interessent keine Informationsunterlagen erhält, so ist das nicht gut für das Geschäft, aber es entsteht kein direkter finanzieller Schaden.
3. Die Wochenstatistik für das Management wird der Risikoklasse C zugeordnet, da ein Fehler in der Statistik keine negative Außenwirkung hat.[2]

Für jede der drei definierten Risikoklassen werden in Tab. 6.3 die notwendigen Qualitätssicherungsmaßnahmen festgelegt, wie z. B. der Nachweis des Testfallüberdeckungsgrades, des Testüberdeckungsgrades oder die Durchführung von Code-Inspektionen.

Die in Abhängigkeit von der Risikoklasse festgelegten Qualitätssicherungsmaßnahmen verringern das Risiko, dass eine gravierende Fehlerwirkung erst im produktiven Betrieb des Programms auftritt.

[2] Die innenpolitische Auswirkung für das Projektteam wird hier nicht bewertet. ☺.

Tab. 6.3 Risikoklassen für Programme – Maßnahmen

Risikoklasse	Qualitätssicherungsmaßnahme/Qualitätssicherungsmaß
A – Hoch	Code-Inspektion durch Review-Sitzung
	Checkliste „Code-Inspektion“[3] 100 % positiv beantwortet
	100 % Testfallüberdeckung
	90 % C1-Testüberdeckung
B – Mittel	Code-Inspektion durch schriftliche Stellungnahme Checkliste „Code-Inspektion“ 100 % positiv beantwortet
	100 % Testfallüberdeckung
	60 % C1-Testüberdeckung
C – Gering	80 % Testfallüberdeckung

6.4 Werkzeuge für die Risikoanalyse

Werkzeuge können die Priorisierung von Testfällen unterstützen. Zum Beispiel kann mit CaseMaker [URL: CaseMaker] jedem Testfall eine Risikoklasse von 1 bis 10 zugeordnet werden. Ein Testszenario erhält dann automatisch die höchste Risikoklasse aller ihm zugeordneten Testfälle. Für die Testplanung lässt sich auf diese Weise eine Übersicht erstellen, in der alle durchzuführenden Testszenarien nach ihrer Priorität geordnet aufgeführt sind und somit die Reihenfolge der Tests festgelegt ist.

Die Einschätzung des Risikos eines Testfalles kann natürlich nur vom Testdesigner und der Fachabteilung vorgenommen werden, nicht von einem Werkzeug.

6.5 Zusammenfassung

- Die Risikoanalyse ist ein wichtiges Instrument der Qualitätssicherung.
- Die ABC-Analyse vereinfacht die Risikoermittlung, indem sie mit drei Risikoklassen statt mit zehn arbeitet.
- Mit der Risikoanalyse werden Testobjekte, Testfälle und Fehler priorisiert, um Risiken von zeit- und kostengetriebenen Entscheidungen beurteilen und QS-Maßnahmen risikobasiert planen zu können.
- Werkzeuge können die Priorisierung von Testfällen unterstützen und den Planungsprozess vereinfachen.

[3] Checkliste „Code-Inspektion“ s. Abschn. 11.1.2

Checklisten 7

Wer in kleinsten Dingen zuverlässig ist, der ist es auch in den großen.

Lukas 16.10

Ein sehr effizientes Hilfsmittel der konstruktiven und analytischen Qualitätssicherung sind wiederverwendbare Checklisten.

7.1 Ziele des Einsatzes von Checklisten

Mit dem Einsatz von standardisierten Checklisten werden mehrere Ziele verfolgt:

1. Checklisten machen Dritten in kompakter Form Erfahrungen zugänglich, die von vielen Personen zusammengetragen wurden.
2. Einheitliche Checklisten stellen sicher, dass bei Prüfungen keine Punkte vergessen werden.
3. Prüfungen, die mit denselben Checklisten durchgeführt werden, sind vergleichbar.
4. Durch Checklisten werden Fehler vermieden, wenn sie als konstruktive Qualitätssicherungsmaßnahme frühzeitig bekannt gemacht und bei der Produktentwicklung berücksichtigt werden.
5. Der geforderte Erfüllungsgrad einer verbindlich vorgeschriebenen Checkliste stellt eine Anforderung dar, an der die Qualität eines zu prüfenden Ergebnisses gemessen werden kann.

Der Erfüllungsgrad einer Checkliste wird im zweiten Teil des Buches, in dem die Testarten beschrieben werden, oft als Maß für ein Qualitätssicherungsmerkmal festgelegt.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_7

Nehmen wir als Beispiel die Checkliste „Code-Inspektion“ aus Abschn. 11.1.2. Eine solche Checkliste wird regelmäßig von den Entwicklern überarbeitet. Sie enthält die aktuellen Programmierstandards sowie die „Tipps & Tricks“ der erfahrenen Programmierer. Sie kann auch auf bestimmte Programmiersprachen (JAVA, .NET, PHP,…) ausgerichtet werden und somit in mehreren spezifischen Ausprägungen eingesetzt werden.

Unerfahrene Entwickler können sich daran orientieren, sie lernen anhand der Checkliste schnell die Standards und Kniffe der betreffenden Programmiersprache kennen.

Weil die Checkliste allen Beteiligten vor Projektbeginn zur Verfügung gestellt wird, werden bestimmte Fehler in der Programmierung von vornherein vermieden.

Mit der Vorgabe, dass für jedes Programm der Risikoklasse A im Rahmen einer Code-Inspektion die Checkliste „Code-Inspektion“ zu 100 % erfüllt sein muss, stellt sie ein Qualitätsmaß dar.

► Der Erfüllungsgrad einer Checkliste ist eine Qualitätsanforderung.

7.2 Werkzeuge für die Checklistenverwaltung

Checklisten können in jedem Text- oder Tabellenkalkulationsprogramm beschrieben werden.

Sollen Checklisten im umfangreichen Maße eingesetzt und die mit ihnen durchgeführten Prüfungen revisionssicher und auswertbar in einer Datenbank abgelegt werden, bieten sich kommerzielle **Reviewmanagement-Werkzeuge** an. Ein checklistenbasiertes Reviewmanagement-Tool ist Q-Chess [URL: QCHESS].

7.3 Empfehlungen zum Einsatz von Checklisten

Um Checklisten effektiv und projektübergreifend einsetzen zu können, sollten sie zentral von einer Qualitätssicherungsinstanz, z. B. dem IT-Qualitätssicherungsverantwortlichen des Unternehmens, verwaltet werden. Diese Instanz muss folgende Aufgaben wahrnehmen:

- Bedarfe für neue Checklisten ermitteln
- Neue Checklisten mit Fachleuten entwerfen
- Checklisten zum Einsatz freigeben
- Bestehende Checklisten regelmäßig überarbeiten
- Veraltete Checklisten aus dem Verkehr ziehen

Damit die Akzeptanz der Checklisten bei allen Beteiligten vorhanden und ihr Nutzen als konstruktives Hilfsmittel möglichst groß ist, sollte der Personenkreis, deren Arbeitsergebnisse mit den Checklisten geprüft werden, an der Erstellung der Checkfragen beteiligt

werden. Zudem sollten die Checklisten, bevor sie zum Einsatz freigegeben werden, von Fachleuten geprüft werden.

Die Formulierung der Fragen einer Checkliste, die als Prüfungsgrundlage dient, ist sehr entscheidend. Die Fragen einer Checkliste sollten so formuliert sein, dass sie mit „ja“ oder „nein“ beantwortet werden können. Allerdings muss in einer Prüfung eine Frage auch mit der Antwort „nicht relevant“ beantwortet werden dürfen.

► Das wesentliche Ziel des Einsatzes einer Checkliste ist der Nachweis, dass über alles nachgedacht wurde, und nicht, dass alle Checkpunkte unbedingt umgesetzt wurden.

Checklisten müssen regelmäßig überarbeitet werden, denn ihre Inhalte können veralten oder Prüfpunkte enthalten, die sich im Einsatz nicht bewährt haben, weil sie zum Beispiel nicht eindeutig sind oder stets mit „nicht relevant“ beantwortet werden.

In diesem Sinne erheben die in diesem Buch aufgeführten Checklisten keinen Anspruch auf Vollständigkeit. Sie werden auch nicht hundertprozentig in jedes Projekt oder auf jedes Prüfobjekt passen. Sie sind aber sicherlich eine gute Basis für die Erstellung von eigenen Checklisten, weil sie kurz und knapp wesentliche Sachverhalte erfassen und einen großen, in der Praxis erworbenen Erfahrungsschatz offen legen.

7.4 Zusammenfassung

- Checklisten unterstützen konstruktive und analytische Qualitätssicherungsmaßnahmen.
- Checkfragen sind so zu stellen, dass sie mit „ja“ oder „nein“ beantwortet werden können. Eine Checkfrage ist auch mit einem begründeten „nicht relevant“ positiv beantwortet.
- Checklisten sind nicht restriktiv einzusetzen, sondern dienen dem Nachweis, an alles gedacht zu haben.
- Eine Qualitätssicherungsinstanz sollte Checklisten an zentraler Stelle verwalten und dafür sorgen, dass sie regelmäßig von Experten überarbeitet werden.
- Mit Reviewmanagement-Werkzeugen können Checklisten und Prüfergebnisse zentral gepflegt, verwaltet und ausgewertet werden.

Teil II
Testarten

8. Prüfungen von Dokumenten
9. Tests zur Funktionalität
10. Tests zur Benutzbarkeit
11. Tests zur Änderbarkeit und Übertragbarkeit
12. Tests zur Effizienz
13. Tests zur Zuverlässigkeit

Der zweite Teil dieses Buches gibt mit der ausführlichen Beschreibung der Testarten dem Tester Methoden, Beispiele, Tipps und Checklisten an die Hand, um die Tests einer Anwendung effizient vorbereiten, durchführen und bewerten zu können.

Jede Testart zeichnet sich dadurch aus, dass sie ein bestimmtes Qualitätsmerkmal überprüft. Die Hauptqualitätsmerkmale von Software sind Funktionalität, Benutzbarkeit, Änderbarkeit, Übertragbarkeit, Effizienz und Zuverlässigkeit, die sich in den einzelnen Kapiteln des zweiten Teils wiederfinden.

Eine Ausnahme ist das Kapitel zur Prüfung von Dokumenten, denn die darin beschriebenen Verfahren prüfen kein Qualitätsmerkmal der zu testenden Software, sondern Merkmale der Dokumente, die diese Software beschreiben. Diese Verfahren haben also „nur“ indirekten Einfluss auf die Qualitätsmerkmale der Software. Die Qualitätssicherung von Dokumenten ist aber sehr wichtig, weil durch sie in frühen Projektphasen konzeptionelle Fehler aufgedeckt werden. Daher beginnt der zweite Teil des Buches mit dem Kapitel über Prüfungen von Dokumenten.

Prüfungen von Dokumenten

8

Wer A sagt, der muss nicht B sagen. Er kann auch erkennen, dass A falsch war.

Bertold Brecht (1898–1956)

Dieses Kapitel behandelt die Qualitätssicherung von Dokumenten. Je nach Inhalt und angesprochener Zielgruppe eines zu prüfenden Dokumentes stehen bei einer Prüfung unterschiedliche Qualitätsmerkmale im Fokus, z. B. Funktionalität bei einer Funktionsbeschreibung, Effizienz bei einem systemtechnischen Konzept und Benutzbarkeit bei einem Anwenderhandbuch. Das bedeutet, dass der Dokumententest als Testart per se keinem bestimmten Software-Qualitätsmerkmal zugeordnet werden kann.

8.1 Dokumententest

► Der Dokumententest überprüft die formale und inhaltliche Vollständigkeit, Widerspruchsfreiheit und Richtigkeit der in einem Dokument beschriebenen Aussagen und Anforderungen.

In einem Software-Entwicklungsprojekt fallen **prozessbezogene** und **produktbezogene Dokumente** an, die einer Prüfung unterliegen können.

Dokumente, die sich auf den Entwicklungsprozess beziehen, sind zum Beispiel Projektmanagementpläne, Statusberichte, Protokolle, und Testkonzepte, die vom Projekt- und Qualitätsmanagement erstellt werden.

Zu den produktbezogenen Dokumenten gehören fachliche und technische Anforderungen sowie Dokumentationen für den Benutzer.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_8

Fachliche Anforderungsdokumente
Fachliche Anforderungsdokumente sind unter anderem Beschreibungen von Algorithmen und Anwendungsfällen (Use Cases), Datenmodelle, Entwürfe von Websites, Funktionsbeschreibungen, Geschäftsprozessmodelle, Objektmodelle, Online- und Druckformulare sowie Testspezifikationen.

Technische Anforderungsdokumente
Zu den technischen Anforderungsdokumenten gehören Datenbank-Design, Sicherheitskonzept, Netzwerk-Design, Rechnerarchitektur, Konzept zur Ausfallsicherheit und Systemressourcenplanung (Zeitverhalten, Speicherbedarf).

Benutzerdokumentation
Zur Inbetriebnahme einer Anwendung werden Benutzerhandbücher, Installationsanweisungen, Online-Hilfen und Schulungsunterlagen benötigt, die nicht ungeprüft freigegeben werden dürfen.

Diese exemplarische Aufzählung von Dokumenten lässt erahnen, wie aufwändig die Qualitätssicherung aller Dokumente in einem IT-Projekt ist.[1] Aber weil der Dokumententest sicherstellt, dass die fachlichen und technischen Anforderungen des Auftraggebers an eine Software richtig verstanden und realisiert werden und weil vollständig und interpretationsfrei beschriebene Anforderungen den Grundstein für ein erfolgreiches Projekt legen, darf der Dokumententest in keinem IT-Projekt vernachlässigt werden. Dokumententests werden in Form von Reviews durchgeführt.

► Ein **Review** ist die geplante, systematische und kritische Prüfung von Arbeitsergebnissen.

Ein Review dient der frühzeitigen Erkennung von Problemen, Fehlern und Inkonsistenzen. Mit einem Review kann auch die Abnahmereife eines Produktes durch Fachleute festgestellt werden. Die Ergebnisse eines Reviews sind zu bewerten und zu protokollieren. Reviews können je nach Prüfziel, Prüfbedingungen und Prüfobjekt unterschiedliche Ausprägungen haben.

Ein bewährtes Vorgehen bei der Qualitätssicherung von Dokumenten besteht darin, jedes fertiggestellte, prüfungsrelevante Dokument zuerst einer formalen Prüfung zu unterziehen. Anschließend wird es in einer Gruppensitzung (Review-Sitzung) durch mehrere Experten geprüft *oder* in einer schriftlichen Stellungnahme durch einen oder mehrere Experten begutachtet.

Im Folgenden wird eine pragmatische Vorgehensweise zur Prüfung von Dokumenten beschrieben, mit der ich positive gemacht habe. Detaillierte Abstufungen von Review-Arten werden hier nicht vorgenommen. Sie sind zum Beispiel in [Spillner_2012] nachzulesen.

[1] Was wohl mit ein Grund dafür ist, dass sie oftmals vernachlässigt wird.

8.1.1 Formale Prüfung

Ein formal korrektes Dokument erspart bei der anschließenden Prüfung des fachlichen Inhalts wertvolle Zeit, denn Diskussionen über Formalitäten müssen in Review-Sitzungen nicht mehr geführt werden. Die Zeit der Fachabteilung ist immer knapp und teuer! Ich habe Reviews beiwohnen müssen, in denen es erst in der dritten Sitzung um Inhalte ging. Zuvor wurden nur endlos „Form und Farbe“ des Dokumentes diskutiert. Eine formale Prüfung kann dokumentenbezogen und/oder inhaltsbezogen sein.

Dokumentenbezogen bedeutet, dass das Dokument selbst auf Einhaltung von Formalismen wie Layout und Gliederung geprüft wird.

Inhaltsbezogen heißt, dass die Darstellung der Inhalte bestimmten Richtlinien und Standards unterliegt, wie zum Beispiel die Grafik eines Objektmodells oder die beschreibenden Eigenschaften eines Anwendungsfalls.

► Die **formale Prüfung** eines Dokumentes stellt die Einhaltung von Standards, Richtlinien und Methoden für das Dokument selbst und für die darin beschriebenen Inhalte sicher.

Für die formale Prüfung von Dokumenten eignen sich Checklisten, wie die beiden folgenden Beispiele zeigen. Die dokumentenbezogene Checkliste „Dokument“ legt die formalen Eigenschaften fest, die jedes Arbeitsergebnis, das in Papierform erstellt wird, erfüllen muss.

✓ Checkliste Dokument

- Hat das Dokument eine eindeutige Identifikation?
- Ist ein Deckblatt vorhanden und ist es vollständig ausgefüllt?
- Ist der Autor genannt?
- Sind Vertraulichkeitsstufen und Urheberrechte ausgewiesen?
- Entspricht das Layout den vorgegebenen Standards?
- Sind Seitenzahlen und Gesamtseiten auf jeder Seite angegeben?
- Ist ein Inhaltsverzeichnis vorhanden?
- Stimmt das Inhaltsverzeichnis mit der Gliederung des Dokumentes überein?
- Existiert ein Abbildungsverzeichnis?
- Existiert ein Tabellenverzeichnis?
- Existiert ein Abkürzungsverzeichnis?
- Existiert ein Quellenverzeichnis?
- Sind alle Referenzdokumente aufgelistet?
- Sind Querverweise eindeutig und richtig?
- Ist das Dokument verständlich gegliedert, übersichtlich aufgebaut und leicht verständlich (Grafiken, Tabellen,…)?
- Ist das Dokument redaktionell vollständig, d. h. fehlen keine Textstellen, Seiten, Abbildungen?
- Ist der Inhalt widerspruchsfrei?

- Sind Begriffe eindeutig definiert und durchgängig verwendet?
- Sind alle nicht allgemein bekannten Begriffe und Abkürzungen definiert?
- Ist das Dokument frei von Rechtschreibfehlern und Grammatikfehlern?
- Ist das Dokument aktuell?
- Ist das Dokument stets mit aktuellem Inhalt reproduzierbar?
- Ist der Status (in Arbeit, erledigt, freigegeben,…) angegeben und richtig?
- Gibt es eine Änderungshistorie?
- Sind die Änderungen markiert?
- Hat das Dokument einen Versionsstand?
- Ist das letzte Speicherdatum des Dokumentes angegeben?

Das zweite Beispiel zur formalen Prüfung eines Dokumentes ist eine inhaltsbezogene Checkliste für statische Objektmodelle. Ein statisches Objektmodell ist der Teil des Informationsmodells, der ohne die dynamischen Aspekte wie Methoden, Sequenzdiagramme und Zustandsübergangsdiagramme modelliert wird.

✓ Checkliste Objektmodell

Allgemein

- Ist die fachliche Beschreibung des Modells korrekt?
- Sind alle im Modell verwendeten Begriffe eindeutig und sind sie mit dem Glossar konsistent?
- Sind keine Prozesse und Prozessabläufe im statischen Teil des Objektmodells modelliert? (Verbergen sich z. B. in den Klassen- und Feldbeschreibungen keine Prozessabläufe?)
- Hat jedes Diagramm eine Beschreibung?
- Ist das Mengengerüst dokumentiert?

Konsistenz zu anderen Modellen

- Ist das Objektmodell konsistent zum Prozessmodell und den Benutzeroberflächen?
- Ist bei den einzelnen Klassen der jeweilige Prozess referenziert, aus dem diese Klasse abgeleitet wurde?
- Finden sich alle relevanten Objekte, Aufgabenbereiche und Rollen, die sich aus dem Prozessmodell ableiten lassen, im Objektmodell wieder?
- Wird jedes fachliche Attribut einer Klasse auf mindestens einer Maske der Benutzeroberflächen verwendet?
- Gilt für jedes Feld auf der Benutzeroberfläche:
  a. es ist das Attribut einer Klasse oder
  b. es ist aus einem oder mehreren Attributen einer oder mehrerer Klassen abgeleitet oder
  c. es ist ein klassenunabhängiges Feld (z. B. Tagesdatum, Zähler auf einer Maske)?

Klassen und Beziehungen

- Hat jede Klasse eine eindeutige Beschreibung, mit Angabe der zugrundeliegenden Objektmenge und möglichst mit Beispiel?
- Bei Generalisierung/Spezialisierung:
  - Lässt sich die Unterklasse jederzeit anstelle der Oberklasse einsetzen, ohne dass der verwendende Prozess etwas davon bemerkt?
  - Sind alle Unterklassen einer Oberklasse nach demselben Unterscheidungskriteriums spezialisiert worden und ist dieses Kriterium sofort ersichtlich?[2]
- Sind die Rollen fachlich richtig angegeben?
- Falls sich Rollen auf Schnittstellen/Umweltobjekte beziehen, existieren diese Rollen dort?
- Bei Komposition: Ist hier fachlich tatsächlich eine Besitz- bzw. Herrschafts-Beziehung gegeben?
- Bei Abhängigkeit: Ist die Abhängigkeit tatsächlich zeitlich begrenzt?
- Bei Assoziation: Gilt die Beziehung tatsächlich permanent?[3]
- Wurden m:n-Assoziationen aufgelöst?
- Sind bei Assoziationen Kardinalitäten angegeben?
- Sind Integritätsbedingungen beschrieben, insbesondere Ableitungsbedingungen und Plausibilitäten?
- Fehler- und Prüfhinweise: Sind keine Textbausteinklassen im Diagramm modelliert sondern separat beschrieben?

Attribute

- Sind für alle Klassen alle Attribute angegeben (insbesondere für Schnittstellen)?
- Sind Namenskonventionen für die Attributnamen definiert und eingehalten worden?
- Gibt es jedes Attribut nur einmal, d. h. es sind eindeutige Attributnamen vergeben?
- Sind alle Felder (insbesondere alle Datums-, Betrags- und Kennzeichenfelder) eindeutig am Namen zu erkennen?
- Hat jedes Attribut eine aussagekräftige Beschreibung?
- Hat jedes Attribut eine Optionalität (kann/muss)?
- Sind zu jedem Attribut Datentyp, Länge und Wertebereich/Domain angegeben?

► Erst wenn die formale Qualität eines Dokumentes sichergestellt ist, kann der fachliche Inhalt effizient in einer Review-Sitzungoder einer schriftlichen Stellungnahme geprüft werden.

[2] Für die Oberklasse Fahrzeug wäre eine falsche Unterklassenbildung: Diesel, Benziner, Motorrad. Richtig ist: Auto, Motorrad, Fahrrad.

[3] Wenn nicht, dann ist eine Abhängigkeit statt Assoziation zu modellieren.

8.1.2 Review-Sitzung

► Eine **Review-Sitzung** ist die geplante, systematische und kritische Prüfung eines Arbeitsergebnisses durch Fachleute in einer Gruppensitzung.

Die Durchführung einer Review-Sitzung unterliegt den folgenden Regeln:

1. Für die Durchführung des Reviews ist genau eine, vorab bestimmte Person verantwortlich, i. d. R. eine Person in der Moderatorenrolle.
2. Der Teilnehmerkreis eines Reviews besteht in der Regel aus 4 bis 6 Personen (Autor(en), Fachleute als Prüfer, Moderator)
3. Die zu prüfenden Unterlagen werden rechtzeitig versendet, d. h. 7 bis 10 Tage vor dem Review-Termin.
4. Für die Review-Sitzung müssen sich alle Teilnehmer vorbereiten, d. h. die Dokumente durcharbeiten und Fragen und Mängel vorab schriftlich notieren.
5. Da alle Teilnehmer vorbereitet sind, wird das Dokument in der Review-Sitzung nicht mehr vom Autor vorgestellt, sondern es werden in der Vorbereitung erkannte Probleme und auftretende Fragen angesprochen.
6. Fachliche Diskussionen sind vom Moderator zu unterbinden, für Problemlösungen werden Tätigkeiten im Protokoll aufgenommen. Feststellungen treffen, nicht diskutieren!
7. Eine Review-Sitzung dauert 2 bis 4 h.
8. Eine vorab bestimmte Person führt das Review-Protokoll.
9. Die festgestellten Mängel werden bewertet.[4]
10. Am Ende einer Sitzung ist eine Entscheidung über das Prüfobjekt zu treffen:
    - Abnahme
    - Abnahme nach Mängelbeseitigung ohne Wiederholung des Reviews
    - Abnahme nach Mängelbeseitigung mit Wiederholung des Reviews
    - Ablehnung
11. Das Protokoll wird an alle Teilnehmer versendet und gilt als akzeptiert, wenn innerhalb eines festgelegten Zeitraumes (zum Beispiel einer Woche) keine Widersprüche eingereicht werden.
12. Im Review-Protokoll vermerkte Mängel sind in einer vom Projektleiter festgelegten Zeit zu beheben und von der Qualitätssicherung oder in einer neuen Review-Sitzung zu prüfen.

In der Literatur wird oft gefordert, dass der Protokollant weder der Moderator, noch ein Prüfer, noch der Autor sein darf. Im Projekt ist dieses Vorgehen aus Personalmangel fast nie realisierbar. Rollen innerhalb einer Review-Sitzung können in Personalunion besetzt werden, ohne dass der Erfolg eines Reviews darunter leidet. Warum sollte der Moderator, einer der Prüfer oder der Autor nicht auch das Protokoll führen, das ohnehin von allen Teilnehmern bestätigt werden muss?

[4] Siehe zum Beispiel Tab. 6.1 im Kap. 6, Risikoanalyse.

Eine Review-Sitzung kann in Form einer Webkonferenz durchgeführt werden, sofern es organisatorisch vertretbar ist.

Projekt- und Testmanagement müssen unter Abwägung der bekannten Risiken entscheiden, für welche Dokumente ressourcenintensive Reviews notwendig sind.

Reviews sind aufwändig

Wie sich aus den obigen Regeln erahnen lässt, sind Review-Sitzungen sehr aufwändig.

Ein Rechenbeispiel: Wenn 5 Personen ein Review über die Dauer von 4 h durchführen, für das sie sich je 2,5 h vorbereitet haben, wenn der Moderator für die Einladung, das Protokoll und die Nacharbeiten 8 h aufwendet und wenn der Autor die Mängel in 8 h behebt, dann sind unterm Strich schnell 6 Personentage investiert. Eine eventuell notwendige Wiederholung des Reviews ist dabei nicht einmal berücksichtigt.

Trotzdem: Reviews spielen ihre Kosten mehrfach wieder ein. In Reviews behobene Fehler treten nicht mehr in späteren Projektphasen oder gar im produktiven Betrieb auf – da würden sie richtig teuer. Wichtig ist, dass das Management die Aufwände, die durch Reviews in frühen Projektphasen entstehen, erkennt und die benötigten Ressourcen budgetiert und einplant.

8.1.3 Schriftliche Stellungnahme

Eine budget- und terminfreundliche Alternative zu einer Review-Sitzung ist die schriftliche Stellungnahme.

► Eine **schriftliche Stellungnahme** ist die zu protokollierende, kritische Durchsicht eines Arbeitsergebnisses durch eine oder mehrere Personen mit Fachkompetenz.

Die schriftliche Stellungnahme wird von einer oder mehreren Personen, die voneinander zeitlich und örtlich unabhängig prüfen, durchgeführt und eignet sich in folgenden Situationen:

- Die Prüfergruppe ist so groß, dass sie den Rahmen eines Reviews sprengen würde.
- Die Prüfer können örtlich und/oder zeitlich nicht zusammengeführt werden.
- Die einem Prüfobjekt zugeordnete Risikoklasse sieht „nur“ eine Vieraugenüberprüfung vor.

Wenn mehrere Prüfer an einer Stellungnahme beteiligt sind, müssen die Aktivitäten von einem Qualitätssicherungsverantwortlichen koordiniert werden:

1. Die Prüfer erhalten die zu prüfenden Unterlagen mit Nennung des Prüfzieles und des Rückgabedatums, das 1 bis 2 Wochen nach dem Versandtag liegt.
2. Jeder Prüfer verfasst ein Prüfprotokoll, das er zum Rückgabedatum abliefert.

3. Der Koordinator fasst alle Prüfprotokolle zusammen und entscheidet anhand der Ergebnisse über den Stand des Prüfobjekts (Abnahme, Abnahme nach Mängelbeseitigung, Wiederholung des Prüfung, Ablehnung).
4. Er erstellt ein Gesamtprotokoll mit allen festgestellten Mängeln, Kommentaren und Antworten sowie die getroffene Entscheidung zur Abnahme des Prüfobjektes.
5. Das Protokoll wird an alle Teilnehmer versendet und gilt als akzeptiert, wenn innerhalb einer Woche keine Widersprüche eingereicht werden.

Wenn es um fachlich wegweisende oder projektrelevante Entscheidungen geht, kann eine schriftliche Stellungnahme ein Review nicht ersetzen, denn schriftliche Stellungnahmen bergen die Gefahr, dass sie von den Prüfern nebenher, d. h. nicht gründlich durchgeführt werden. Bei einer Review-Sitzung dagegen wird schnell offensichtlich, wer seine Hausaufgaben gemacht hat.

Ein „gesunder“ Mix aus Reviews und schriftlichen Stellungnahmen zur Prüfung von Dokumenten ist daher jedem Projekt zu empfehlen.

8.2 Spezielle Dokumententests

Es gibt produktbezogene Dokumente, die aufgrund ihres speziellen Inhalts per se einem Qualitätsmerkmal zugeordnet werden können. Ihre Prüfungen finden in speziellen Dokumententests statt, die eigene Testarten darstellen. Diese sind in den entsprechenden Kapiteln beschrieben und der Vollständigkeit halber hier aufgeführt.

Desktop- und Mobile-Websites können als eigenständiges Dokument angesehen werden, dessen Inhalt den Standards und rechtlichen Vorgaben des WWW genügen muss. Die Prüfung des Inhaltes einer Website wird im **Content-Test** mit den Methoden des Dokumententests durchgeführt. Der Content-Test überprüft als Schwerpunkt das Qualitätsmerkmal Benutzbarkeit eines Webauftritts und wird im Kap. 10.1 beschrieben.

Code-Walkthroughs und **Code-Inspektionen** sind spezielle Ausprägungen eines Reviews. Weil sie programmiertechnisches Wissen von den Prüfern verlangen und gezielt die Qualitätsmerkmale Änderbarkeit und Übertragbarkeit prüfen, werden diese beiden Prüfungsarten im Kap. 11.1, Code-Analysen, beschrieben.

8.3 Werkzeuge für den Dokumententest

In IT-Projekten sollte die Projektdokumentation mit **CASE[5]-Tools** erfolgen, denn aus Sicht der Qualitätssicherung bieten sie viele Vorteile:

- Der Einsatz eines CASE-Tools zwingt das Projektteam, nach vorgegebenen Methoden und Vorlagen zu arbeiten.

[5] CASE = Computer Aided Software Engineering.

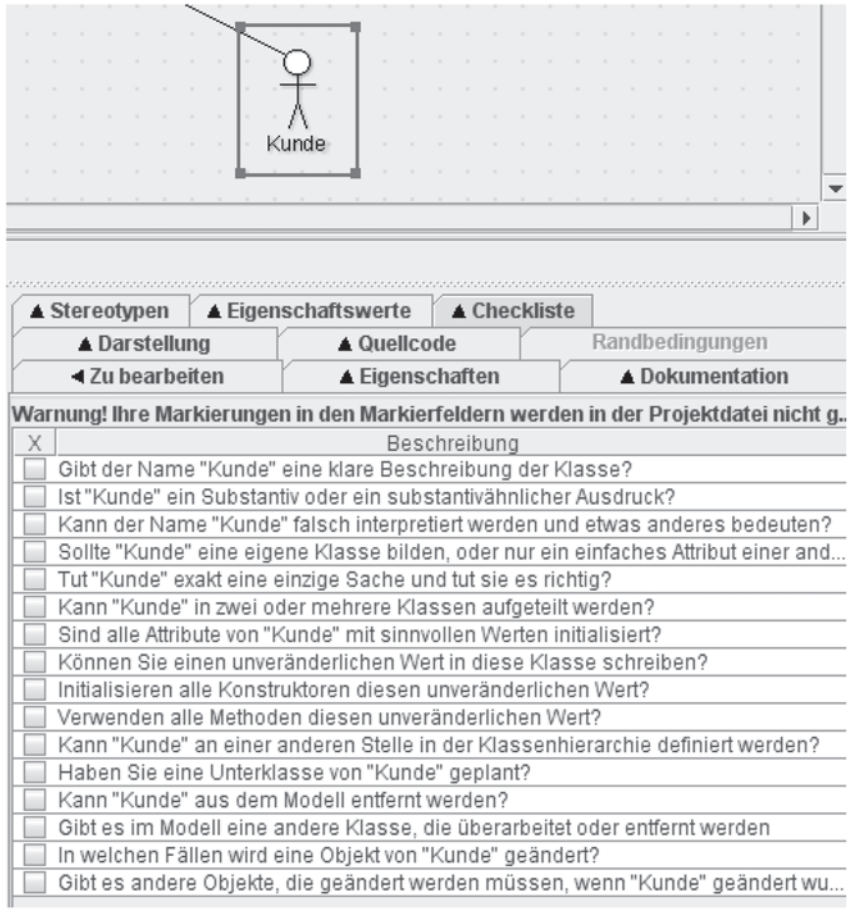


Abb. 8.1 Checkliste <Akteur> in ArgoUML

- Der Aufbau und die Darstellung der Arbeitsergebnisse sind von den einzelnen Autoren unabhängig.
- Die Dokumente werden zentral verwaltet.
- Die Versionsführung der Dokumente ist gewährleistet.
- Formale Prüfungen zur Vollständigkeit und Widerspruchsfreiheit der Dokumente sowie zur Einhaltung von Modellierungsregeln werden automatisch ausgeführt.

Eine etabliertes Open Source Werkzeug zur Darstellung von UML-Diagrammen ist zum Beispiel ArgoUML [URL: ArgoUML]. **ArgoUML** liefert Checklisten für einzelne UML-Elemente mit, wie zum Beispiel für einen „Akteur" eines Use-Case-Diagramms in Abb. 8.1.

Die Grafiken in Abb. 8.1 (Zustandsdiagramm) und 4.3 (Anwendungsfalldiagramm) sind übrigens mit dem Open Source Tool **StarUML** erstellt worden [URL: StarUML].

Unter [URL: CASE-Tools] ist eine umfangreiche Liste von CASE-Tools aufgeführt.

8.4 Qualitätsanforderungen zum Dokumententest

Qualitätsmerkmale, die für jedes abnahmerelevante Dokument nachgewiesen werden müssen, sind formale Korrektheit und Vollständigkeit sowie inhaltliche Fehlerfreiheit, Widerspruchsfreiheit und Vollständigkeit. Externe Dokumente müssen zudem benutzerfreundlich sein.

Die formale, dokumentenbezogene Korrektheit und Vollständigkeit kann anhand der Checkliste „Dokument“ (s. Kap. 8.1.1) gemessen werden. 100 % Überdeckung der Checkliste bedeutet, dass im Rahmen der Prüfung alle Fragen der Checkliste mit „ja“ oder „nicht relevant“ beantwortet wurden.

Für Anforderungsdokumente, die nach speziellen Methoden erstellt werden müssen, sollten eigene Checklisten zur inhaltsbezogenen Prüfung festgelegt werden. Dann kann zum Beispiel gefordert werden, dass für ein Datenmodell die Checkliste „Objektmodell“ (s. Kap. 8.1.1) zu 100 % erfüllt sein muss.

Fachliche Fehlerfreiheit, Widerspruchsfreiheit und Vollständigkeit eines Dokuments werden anhand von Prüfprotokollen nachgewiesen, die bei der Durchführung von Reviews und schriftlichen Stellungnahmen erstellt werden. Die Qualitätsanforderung ist, dass jedes Dokument nachweislich in einem Prüfprotokoll vom Auftraggeber abgenommen wird.

Die Benutzerfreundlichkeit von Dokumenten mit Außenwirkung (Benutzerhandbücher, Installationsanweisungen,…) kann durch Testnutzer festgestellt werden. Als Qualitätsanforderung kann zum Beispiel festgelegt werden, dass ungeschulte Nutzer, die ein Dokument lesen, im Durchschnitt höchstens vier Verständnisfragen pro 50 Seiten Text stellen dürfen.

8.5 Empfehlungen zum Dokumententest

Dokumententests in frühen Phasen

Der Dokumententest hat seine Hoch-Zeit in den frühen Phasen eines Projektes, wenn die meisten konzeptionellen Arbeitsergebnisse entstehen. Da Reviews und schriftliche Stellungnahmen sehr ressourcenaufwendig sind, müssen sie rechtzeitig zum Projektbeginn geplant und budgetiert werden. Eine Balance zwischen Termin, Budget und Qualität sollte dabei erreicht werden.

Weil durch Dokumententests viele Fehler früh aufgedeckt werden, wirken sie sich positiv auf das Projektbudget aus. Und allein dass sie durchgeführt werden, motiviert so manchen Autor von vornherein, qualitativ hochwertige Spezifikationen zu schreiben.

Checklisten für formale Prüfungen

Zu Beginn eines Projektes sollte für jeden zu erstellenden Dokumententyp, der auf die Einhaltung von Regeln und Standards geprüft werden muss, eine Checkliste für die formale Prüfung bereitgestellt werden. Einerseits als konstruktive Qualitätssicherungsmaßnahme, andererseits, um fachliche Prüfungen effizient durchführen zu können.

Vier-Augen-Prinzip
Ein ungeschriebenes Gesetz sollte in jedem Projekt sein, dass jedes Dokument, das für Dritte relevant ist, dem Vier-Augen-Prinzip unterliegt.

Tools einsetzen
Für umfangreichere Projekte empfiehlt sich der Einsatz eines **CASE-Tools**. Er ist unter der Voraussetzung, dass die Mitarbeiter die Methoden und das Tool beherrschen, als konstruktive Qualitätssicherungsmaßnahme sehr wirkungsvoll, weil so formal korrekte, einheitliche und stets aktuell abrufbare Spezifikationsdokumente erzwungen werden.

Auch **Anforderungsmanagementwerkzeuge** (s. [URL: AM-Tools]), die Freigaben auf Basis der einzelnen Anforderungen erlauben, sind sehr gut für den Dokumententest und insbesondere für schriftliche Stellungnahmen geeignet.

8.6 Zusammenfassung

- Wenn Dokumente, an denen sich das Projektgeschehen und die Anwendungsentwicklung orientieren, unvollständig, nicht eindeutig oder gar fehlerhaft sind, wird es sehr teuer, die nicht rechtzeitig erkannten Mängel nachträglich zu beheben.
- Der Dokumententest ist keine Domäne der Qualitätssicherung von Web- oder Mobile-Apps, sondern muss in jedem IT-Projekt eingeplant werden.
- Dokumententests können für projektbezogene und für produktbezogene Dokumente durchgeführt werden.
- Dokumententests werden in Form von Reviews oder schriftlichen Stellungnahmen durchgeführt, denen eine formale Prüfung vorausgeht.
- Für die formale Prüfung von Dokumenten eignen sich Checklisten, die vor der Erstellung der Dokumente den Autoren als konstruktive Qualitätsmaßnahme bereitgestellt werden.
- Reviews und schriftliche Stellungnahmen unterliegen festen Regeln.
- Die Prüfergebnisse eines Dokumententests werden in einem Prüfprotokoll festgehalten.
- Der Einsatz eines CASE-Tools ist eine wichtige konstruktive Qualitätssicherungsmaßnahme.
- CASE-Tools können formale Prüfungen übernehmen.
- Content-Tests, Code-Walkthroughs und Code-Inspektionen werden mit den statischen Methoden des Dokumententests durchgeführt.
- Das Vier-Augen-Prinzip gilt für alle Dokumente, die an Dritte gegeben werden.

Tests zur Funktionalität

9

Verantwortlich ist man nicht nur für das, was man tut, sondern auch für das, was man nicht tut.

Laotse, 6. Jh. v. Chr.

In diesem Kapitel werden die Testarten beschrieben, die das Qualitätsmerkmal Funktionalität im Fokus haben. Sie stellen sicher, dass die von einer Anwendung geforderten Funktionen angemessen, vollständig und korrekt realisiert worden sind, dass die einzelnen Komponenten zu einem fehlerfrei funktionierenden Gesamtsystem integriert werden können und dass die Sicherheit der Daten und des Systems gewährleistet ist.

Die Testarten Unit-Test, funktionaler Komponententest, Integrationstest und funktionaler Systemtest kommen normalerweise in jedem Anwendungsentwicklungsprojekt[1] zum Einsatz.

Zusätzlich werden die Testarten betrachtet, die im Umfeld von Web- und Mobile-Technologien berücksichtigt werden müssen. Dabei handelt es sich um den Link-Test, Cookie-Test, Plugin-Test, Test der Browser-Einstellungen und Webservice-Test. Dazu kommen der Sicherheitstest, der Interoperabilitätstest und der Mobile-Funktionstest.

9.1 Unit-Test

Bevor der Unit-Test beschrieben wird, wollen wir uns vom Begriff **Entwicklertest** verabschieden. Er beschreibt eine personenbezogene Sicht auf das Testen, nämlich alle Testaktivitäten, die von den Entwicklern durchgeführt werden, ohne dass eine Aussage über

[1] Die Testarten sind nicht mit den zum Teil gleichnamigen Teststufen zu verwechseln (s. Kap. 16).

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_9

die zu testenden Objekte getroffen wird. Demnach ist der Entwicklertest keine Testart (s. Kap. 3.1).[2]

Oft wird der Unit-Test als Entwicklertest bezeichnet, in [ISTQB_2013] werden Unit-Test und Komponententest gleichgesetzt und in agilen Projekten wird nur von Unit-Test, nicht von Entwickler- oder Komponententest gesprochen.

…Ja was denn nun?

9.1.1 Was ist ein Unit-Test?

Eine **Komponente** – englisch **Unit** – ist die kleinste Software-Einheit, für die eine Spezifikation verfügbar ist und die isoliert getestet werden kann.

Die kleinste zu testende Einheit im objektorientierten Umfeld ist aus Entwicklersicht eine Methode oder eine Klasse. Aus der Sicht eines Testers, der i. d. R. keine einzelnen Klassen testet, ist es eine Software-Komponente, die mehrere Klassen enthalten kann.

Soweit zur Unit, aber was ist ein Unit-Test? Hier die Definition des Unit-Tests von Frank Westphal [Westphal_2005]:

► „Eine Art von Tests, die wir zeitnah zur Programmierung erstellen und nach jeder Programmmodifikation ausführen wollen, sind **Unit-Tests**. Im Unit-Test werden kleinere Programmteile in Isolation von anderen Programmteilen getestet. Die Granularität der unabhängig getesteten Einheit kann dabei von einzelnen Methoden über Klassen bis hin zu Komponenten reichen. Unit-Tests sind in den überwiegenden Fällen White-Box-Tests. Das bedeutet, der Test kennt die Implementierungsdetails seines Prüflings und macht von diesem Wissen Gebrauch.“

Je nach Sicht des Testenden hat eine Unit (alias Komponente) eine andere Granularität und eine andere „Blickdichte“. Der Entwickler hat Methoden, Klassen und miteinander kommunizierende Klassen (Pakete) im Fokus; den Tester interessieren Komponenten aus fachlich funktionaler Sicht. Der Entwickler führt im Wesentlichen Whitebox-Tests durch; der Tester hat seinen Schwerpunkt auf Blackbox-Tests.

Aus diesen Gründen werden hier Unit-Test und funktionale Komponententest (s. Kap. 9.2) jeweils als eigene Testarten betrachtet.

9.1.2 Durchführung von Unit-Tests

Beim Unit-Test wird für jede Klasse eine eigene Testklasse kodiert, mit der möglichst alle Methoden der zu testenden Klasse aufgerufen und überprüft werden können. Jeder Son-

[2] Dementsprechend müssten alle anderen Tests „Testertests“ genannt werden. ☺

der- oder Fehlerfall (jede eingehende und jede ausgehende Exception) einer Methode oder eines Objekts wird als eigener Testfall implementiert.

Für den Entwickler bedeutet das, dass er sein Programm sukzessive anhand der Ein- und Ausgabeanforderungen entwickelt und testet. Er programmiert einige Funktionalitäten, beschreibt die dazugehörenden Testfälle und führt die Tests durch. Dabei prüft er die internen Programmstrukturen. Sind die Tests erfolgreich, werden Testfälle und Code so lange in kleinen Schritten weiterentwickelt, bis der komplette Code programmiert ist. Bei jedem neuen Schritt werden die bestehenden Testfälle wieder mitgetestet.

Nachdem eine Klasse programmiert und getestet ist, wird sie mit fachlich und/oder technisch zusammengehörenden Klassen zu Paketen (Packages) integriert. Dazu werden Vererbung und Nachrichtenaustausch zwischen dienstanbietenden Objekten auf dem Server und dienstnutzenden Objekten auf dem Client getestet.

Wenn *vor* jedem Codierungsschritt der zugehörenden Testfall spezifiziert und anhand dessen der Code entwickelt wird, wird von testgetriebener Programmierung (**Test Driven Development**) gesprochen.

9.1.3 Abgrenzung und Empfehlung zum Unit-Test

Das beschriebene Vorgehen wirft einige Fragen auf:

- Sind Unit-Tests aufwändig?

  Nein. Wenn die Entwicklung die Tests, die sie ohnehin macht, etwas formaler aufschreibt und Unit-Test-Frameworks (s. Abschn. 9.1.4) nutzt, dann sind die Unit-Tests nicht aufwändig und sie sichern die Tests sogar automatisiert wiederholbar.

- Wenn die Entwicklung nun schon Unit-Tests nachweislich durchführt, ist dann überhaupt noch ein nachgelagerter funktionaler Komponententest notwendig?

  Nein, wenn sichergestellt ist, dass die Entwicklung alle funktionalen Testfälle definiert und durchführt, die für eine vollständige Verifikation der Anforderungen notwendig sind.

- Müssen auf Komponentenebene die Rollen Entwickler und Tester getrennt werden?

  Nein, wenn sichergestellt ist, dass das Projektteam alle funktionalen Testfälle definiert und durchführt, die für eine vollständige Verifikation der Anforderungen notwendig sind. Genau das ist ein Ziel der agilen Software-Entwicklung.

- Können Testteam und Entwicklungsteam effizient zusammen testen, unabhängig vom Vorgehensmodell?

Ja, mit folgender Vorgehensweise:

1. Eine Person mit Testkompetenz definiert parallel zur Spezifikationserstellung mit Blackbox-Verfahren die Testfälle für eine zu testende Komponente und stellt sie dem Entwicklungsteam zur Verfügung.
2. Die Entwicklung führt mit den bereitgestellten Testfällen parallel zur Programmierung ihre Unit-Tests durch, die mit geeigneten Testframeworks (s. Abschn. 9.1.4) automatisiert und dokumentiert werden.
3. Die Komponente wird nach dem Unit-Test an das Testmanagement übergeben. Anhand der festgelegten Qualitätsanforderungen wird entschieden, ob die Testnachweise der Entwicklung hinreichend sind oder ob und welche Testfälle ergänzend vom Testteam durchgeführt werden müssen.

Dieses Vorgehen bringt einige Synergien mit sich. Das Entwicklerteam muss sich keine eigenen Testfälle ausdenken, Mängel in den Anforderungsspezifikationen und Testfällen werden frühzeitig entdeckt und ein Teil des Testaufwands wird in eine frühe Projektphase verlagert. Sollte ein Komponententest nachgelagert sein, so muss das Testteam „nur“ noch die nicht im Unit-Test ausgeführten Testfälle testen.

9.1.4 Werkzeuge für den Unit-Test

Programmierung und Test sind beim Unit-Test eng miteinander verbunden, denn mit jeder Erweiterung des Codes müssen alle bisher durchgeführten Tests wiederholt werden. Das ist ohne entsprechende Werkzeuge nicht durchführbar. Daher werden **Testframeworks** eingesetzt. Mit ihnen werden unter kontrollierten Bedingungen die Tests automatisch wiederholt. Programm- und Testdokumentation werden auf diese Weise stets aktuell fortgeschrieben. Zudem werden die Testergebnisse mit einem Testframework protokolliert und ausgewertet.

Testframeworks gibt es für fast jede Programmiersprache. Eine der bekanntesten Frameworks ist das Java-Framework **JUnit** [URL: junit].

TestNG ist ein Framework für Java-Tests, das sich in den letzten Jahren einen Namen gemacht hat, es basiert auf JUnit und NUnit [URL: TestNG].

Eine Aufstellung von Testframeworks ist in [URL: TestTools OPENSOURCE] unter der Rubrik „Unit testing tools“ aufgeführt.

9.1.5 Qualitätsanforderungen zum Unit-Test

Wenn die Tests parallel zur Programmierung und werkzeugunterstützt im Testframework durchgeführt werden, sind alle Testergebnisse nachweislich und ohne großen Aufwand für den Entwickler dokumentiert. Und weil zu jeder programmierten Methode eine zugehö-

rende Testmethode kodiert ist, ist automatisch die Testmetrik der **Methodenüberdeckung** zu 100 % erfüllt, die für den Unit-Test minimal eingefordert werden sollte.

Ansonsten leiten sich Qualitätsanforderungen zu Testfallüberdeckungsgraden und Testüberdeckungsgraden von den Box-Verfahren ab, mit denen die Testfälle für Unit-Tests entworfen wurden (s. Abschn. 4.1.7 und 4.2.4).

9.2 Funktionaler Komponententest[3]

Es soll noch Entwickler geben, die nicht (gerne) testen. Es soll noch Software geben, die nicht objektorientiert entwickelt wird. Es soll noch Projekte geben, die nicht agil vorgehen. Es soll noch Projekte geben, bei denen der Unit-Test nicht alle definierten Grenzwerte, Statusübergänge und kombinatorischen Testfälle abdeckt. In diesen Fällen muss ein Komponententest durchgeführt werden (Abb. 9.1).

► Im **funktionalen Komponententest** wird eine einzelne Software-Komponente mit dem Ziel getestet, Fehler aufzudecken und die korrekte und vollständige Umsetzung der funktionalen Anforderungen an diese Software-Komponente nachzuweisen.

Beispiel einer Komponente

Betrachten wir als Beispiel den Finanzierungsrechner aus Kap. 4.1. Der Finanzierungsrechner ist eine separat zu testende Software-Komponente, weil für ihn eine Spezifikation vorliegt, er eine funktionale Einheit bildet und nicht sinnvoll unterteilt werden kann.

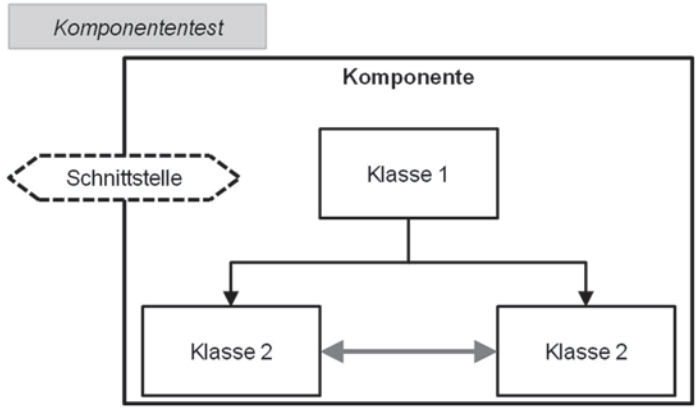


Abb. 9.1 Komponententest

[3] Im Folgenden wird die Testart funktionaler Komponententest beschrieben, der die geforderte Funktionalität einer Software-Komponente sicherstellt und nicht mit der Teststufe Komponententest des V-Modells (s. Kap. 16.1) zu verwechseln ist.

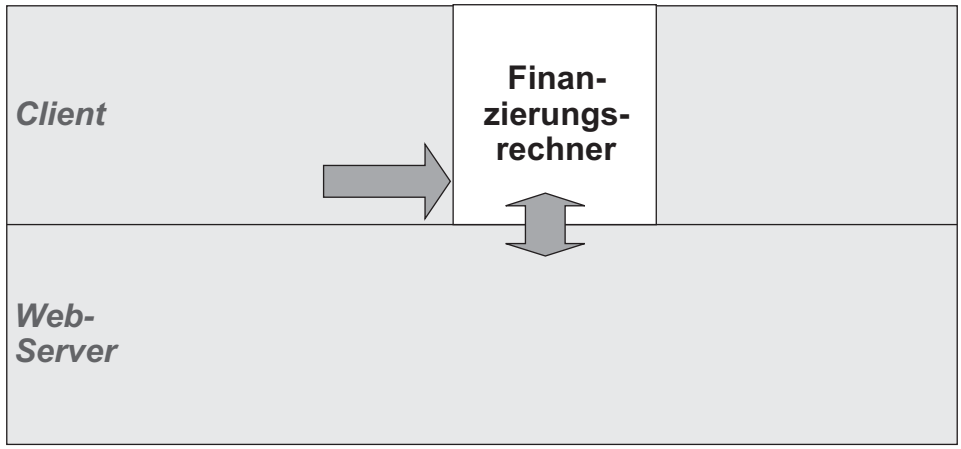


Abb. 9.2 Komponente Finanzierungsrechner

Die Programmlogik des Finanzierungsrechners wird zum Teil auf dem Client und zum Teil auf dem Server durchgeführt (s. Abb. 9.2). Die Plausibilitätsprüfungen der Benutzereingaben werden auf dem Client vorgenommen und die Berechnung der Finanzierung auf dem Server. Für den Komponententest des Finanzierungsrechners wird die Schnittstelle zum Autokonfigurator durch ein Treiberprogramm ersetzt. Die Schnittstelle zur Verfügbarkeitsanfrage an die Datenbank übernimmt ein Platzhalter.

9.2.1 Schritte des Komponententests

Die Funktionalität, die in einer Komponente realisiert ist, ist aus fachlicher Sicht in sich abgeschlossen und wird im Komponententest in mehreren Schritten getestet:

1. Blackbox-Tests:
   Im ersten Schritt werden die mit Blackbox-Verfahren (s. Kap. 4.1) entworfenen Testfälle in einem Blackbox-Test ausgeführt.
2. Whitebox-Tests:
   Nachdem die Testfälle des Blackbox-Tests erfolgreich durchgeführt worden sind, d. h. durch sie keine Fehlerwirkungen mehr aufgedeckt werden, werden Whitebox-Tests für die Software-Komponente durchgeführt. Dieser zweite Schritt ist nur notwendig, wenn in den Qualitätsanforderungen der Nachweis von Testüberdeckungsgraden gefordert ist (s. Kap. 4.2).
3. Erfahrungsbasierte Tests:
   Bewährt hat sich als zusätzlicher Test der Fehlererwartungstest, der eine Software-Komponente auf Basis intuitiv erstellter Testfälle (s. Kap. 4.3) testet.

9.2.2 Werkzeuge für den Komponententest

Um die Schnittstellen einer Komponente mit Testdaten bedienen zu können, müssen ggf. Testtreiber und Platzhalterprogramme entwickelt werden. **Testtreiber** rufen die zu testende Komponente auf, **Platzhalter** werden von der zu testenden Komponente aufgerufen.

Als „intelligente" Treiber bzw. Platzhalter werden auch Mocks eingesetzt, die nicht nur beim Entwicklertest zum Einsatz kommen. Soll die Interaktion eines Objektes mit seiner Umgebung getestet werden, muss diese Umgebung nachgebildet werden. Das kann umständlich, zeitaufwändig, eingeschränkt oder überhaupt nicht möglich sein. Hier kommen Mocks ins Spiel. Ein **Mock** simuliert die Schnittstelle, über die das zu testende Objekt auf seine Umgebung zugreift. Dabei stellt es sicher, dass die erwarteten Methodenaufrufe vollständig, mit den korrekten Parametern und in der richtigen Reihenfolge durchgeführt werden, und es gibt zuvor festgelegte Werte an das Testobjekt zurück. Für die Realisierung von Mocks gibt es **Mocking-Frameworks**, wie z. B. **Mockito**, das ursprünglich für Java und mittlerweile auch für andere Sprachen entwickelt wurde [URL: Mockito].

Die im Abschn. 9.1.4 beschriebenen **xUnit-Tools** sind per Definition für den Komponententest relevant.

Für die Ausführung der Tests können **Capture Replay Tools**[4] eingesetzt werden. Damit wird die Durchführung von Testszenarien aufgezeichnet und wieder abgespielt. Obwohl Capture Replay Tools ursprünglich für die Automatisierung von Regressionstest entwickelt worden sind, sind sie für den Komponententest sehr interessant. Mit ihnen können nicht nur Tests aufgezeichnet und wieder abgespielt, sondern ganze Testläufe mittels Skriptsprachen programmiert werden.

Testüberdeckungsgrade werden, sofern gefordert, im Rahmen des Komponententests nachgewiesen. Dazu sind **Code Coverage Tools** notwendig, deren Arbeitsweise im Abschn. 4.2.3. beschrieben ist.

9.2.3 Qualitätsanforderungen zum Komponententest

Die im Rahmen des Komponententests auszuführenden Testfälle werden mit den Blackbox- und Whitebox-Verfahren entworfen. Daher leiten sich die Qualitätsanforderungen zum Komponententest wie Testfallüberdeckungsgrad und Testüberdeckungsgrad von den Box-Verfahren ab. Sie sind ausführlich in den Abschn. 4.1.7 und 4.2.4 beschrieben.

9.3 Integrationstest

Um eine (Web-)Anwendung betreiben zu können, müssen die Systemkomponenten zu einem Gesamtsystem integriert werden. Dazu werden im Integrationstest die einzelnen Komponenten sukzessive zusammengeführt, bis in der letzten Stufe der Integration das

[4] S. Kap. 14.3, Werkzeuge für die Testwiederholung.

gesamte System zur Verfügung steht und im Systemtest komplette Geschäftsvorfälle durchlaufen und getestet werden können.

► Der **Integrationstest** prüft das korrekte funktionale und technische Zusammenspiel von Hard- und Software-Komponenten, mit dem Ziel, Fehlerwirkungen in den Schnittstellen und der Kommunikation zwischen den Komponenten aufzudecken.

Als Voraussetzung für den Integrationstest müssen die zu integrierenden Komponenten zuvor geprüft worden sein und fehlerfrei funktionieren.

Für jede Abhängigkeit zwischen zwei Komponenten werden für den Integrationstest Testfälle definiert, die nachweisen, dass die Komponenten über ihre Schnittstellen korrekt kommunizieren, d. h. Daten austauschen und gemäß den Anforderungen richtig funktionieren. Zusätzliches Augenmerk liegt beim Integrationstest auf der Fehlerbehandlung von Ausnahmesituationen.

9.3.1 Strategien der Integration

Es gibt unterschiedliche Strategien, wie Komponenten zu einem Gesamtsystem integriert werden können.

Bei der **Backbone-Integration** werden fertiggestellte Komponenten in einen zuvor erstellten Testrahmen aus Treibern, Platzhaltern und Mocks in der zeitlichen Reihenfolge ihrer Fertigstellung eingeklinkt.

Werden die Komponenten in der zeitlichen Reihenfolge ihrer Fertigstellung zusammengefügt, ohne einen Testrahmen zur Verfügung zu haben, wird von **Ad-hoc-Integration** gesprochen. Die Tests können bei diesem Vorgehen zeitlich nicht optimiert durchgeführt werden, dafür entfällt der Aufwand für die Erstellung eines kompletten Testrahmens. Allerdings müssen bei Bedarf auch Treiber und Platzhalter oder Mocks programmiert werden.

Integrationsverfahren wie die strenge **Top-down**- oder **Bottom-up-Zusammenführung** von Modulen sind in der flachen Programmhierarchie von Web- und Mobile-Apps ungeeignet.

Ein „**Big Bang**“, der alle Komponenten gleichzeitig zusammenführt und testet, verbietet sich von selbst. Denn wie soll mit vertretbarem Aufwand der verursachende Fehlerzustand einer auftretenden Fehlerwirkung in einer komplexen Webapplikation lokalisiert werden?

Bewährt hat sich die **anwendungsfallorientierte Integration** der einzelnen Komponenten anhand von Anwendungsfällen und Geschäftsvorfällen. Aber auch für dieses Vorgehen müssen Treiber und Platzhalter bereitgestellt werden.

In der Praxis lassen sich die beschriebenen Integrationsstrategien nicht scharf trennen, sondern in der Regel werden Mischformen umgesetzt.

In der agilen Software-Entwicklung setzt sich die kontinuierliche Integration durch (engl. **Continuous Integration**, CI). Bei diesem Prozess checkt jeder Entwickler am bes-

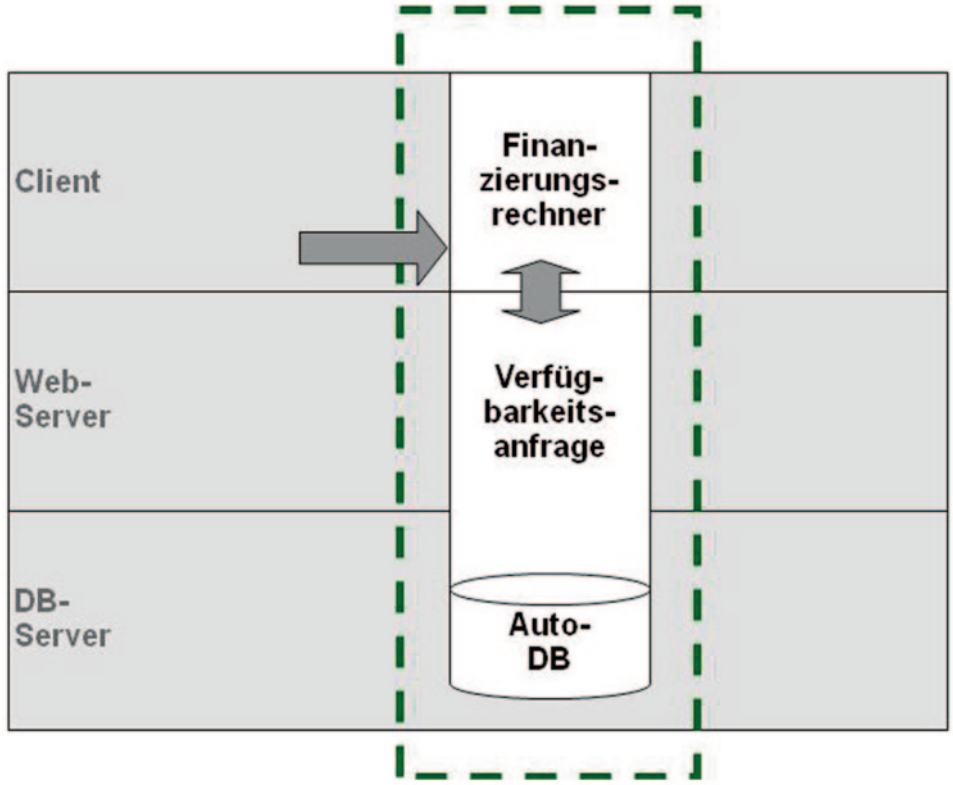


Abb. 9.3 Integration Verfügbarkeitsanfrage

ten täglich – eben kontinuierlich – seine neuen oder geänderten Code-Bausteine in ein Versionsverwaltungssystem ein. Sobald das passiert ist, wird automatisch das komplette System neu kompiliert, einer statischen Code-Analyse unterzogen und in die Testumgebung übertragen. In dieser laufen dann automatisch Unit-Tests, Integrationstests und Systemtests ab. Auf diese Weise steht immer ein lauffähiges System zur Verfügung, sofern der CI-Prozess keine Fehler aufdeckt.

Beispiel zum Integrationstest

Nehmen wir unser Beispiel des Finanzierungsrechners wieder auf.

1. **Integrationsstufe:**

Die Komponente <Finanzierungsrechner> (s. Abb. 9.2) ist vom Testteam im Komponententest nach allen Regeln der Testkunst geprüft worden und wird in den Integrationstest gegeben, damit sie mit der Komponente <Verfügbarkeitsanfrage> zusammengeführt werden kann (s. Abb. 9.3).

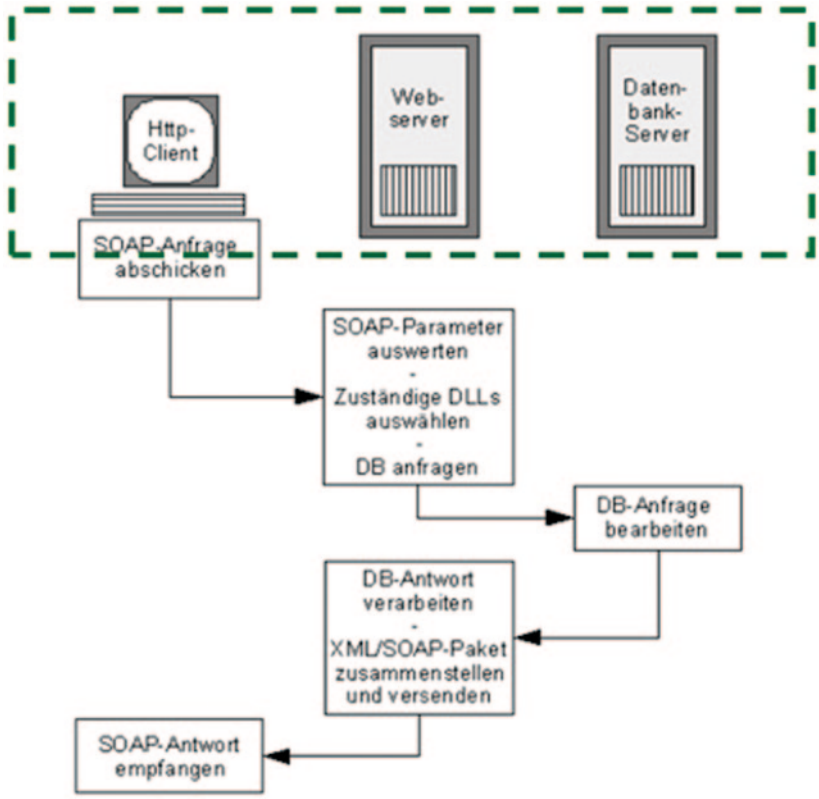


Abb. 9.4 SOAP-Anfrage

Wenn der Kunde mit dem Finanzierungsrechner eine Finanzierungsmöglichkeit berechnet hat und ein verbindliches Angebot haben möchte, muss eine Anfrage an die produktive Datenbank gestellt werden. Diese stellt fest, ob und ab wann das konfigurierte Fahrzeug zur Verfügung steht. Die Verfügbarkeitsanfrage enthält selbst keine Verarbeitungslogik.

In dieser Integrationsphase muss die Anbindung der Datenbank mit den dazu notwendigen Kommunikationskomponenten geprüft werden. In unserem Beispiel ist die Kommunikation der Systemkomponenten mit SOAP[GL] realisiert. Die beteiligten Komponenten und die Kommunikationswege sind in Abb. 9.4 skizziert.

Der Benutzer stellt seine Anfrage über seinen Internetzugang (HTTP-Verbindung über Port 80), die über einen Webserver an eine Datenbank weitergeleitet und von dort auf demselben Weg zurückgegeben wird.

Beim Integrationstest wird überprüft, ob die benötigten Systemkomponenten (Client, Webserver, Datenbank-Server, Kommunikations-DLLs) zur Anfrage der Verfügbarkeit fehlerfrei zusammenarbeiten. Dazu müssen fachliche und technische Testfälle entworfen werden.

Fachliche Testfälle
Aus fachlicher Sicht gibt es zwei Testfälle, die nach der Äquivalenzklassenanalyse (s. Abschn. 4.1.1) gebildet werden:

1. Das konfigurierte Fahrzeug ist laut DB-Anfrage verfügbar.
2. Das konfigurierte Fahrzeug ist nicht verfügbar.

Technische Testfälle
Aus technischer Sicht müssen mehrere Situationen getestet werden. Tritt irgendwo auf dem Weg der Nachrichtenübermittlung ein Fehler auf, muss einerseits der Benutzer eine hilfreiche Meldung im Browser erhalten und anderseits der Systemadministrator benachrichtigt werden. Für jede Fehlersituation, die während des Betriebes auftreten kann, wird ein Testfall benötigt. Exemplarisch ist hier der Testfall beschrieben, der die Reaktion auf eine nicht bereitstehende DLL überprüft:

Testfall: Verteiler-DLL auf Webserver nicht verfügbar

Beschreibung:
Der Webserver wertet die Anfrage des Web-Clients aus und ruft als Ergebnis die für den Datenbank-Zugriff zuständige DLL auf. Steht diese DLL dem Webserver nicht zur Verfügung, muss diese Situation korrekt abgefangen und eine Meldung an den Benutzer gegeben werden.
Erwartete Ergebnisse:

1. Internal Server Error, HTTP-Fehler 500, Rückgabe einer SOAP-Message im Response Body mit dem SOAP-Fault: „Leider liegt zur Zeit ein technisches Problem vor, bitte versuchen Sie es später noch ein Mal."
2. Eintrag in eine Protokolldatei für den Systemadministrator: HTTP-Fehler 500 – Internal Server Error, inkl. SOAP-Message wie oben.

Weitere technische Fehlersituationen, die in Testfällen zu behandeln sind, lassen sich für unser Szenario schnell finden:

- Der Webserver steht nicht bereit.
- Der Datenbank-Server steht nicht bereit.
- Die Auto-Datenbank steht nicht zur Verfügung.

Für unseren Finanzierungsrechner haben wir nun nach der Integration der Datenbankanfrage die in Abb. 9.3 dargestellte Integrationsstufe erreicht.

2. **Integrationsstufe:**
Bisher wird für den Test ein Treibermodul eingesetzt, dass dem Finanzierungsrechner den Kaufpreis des Fahrzeugs übergibt. Im nächsten Schritt soll der Finanzierungsrech-

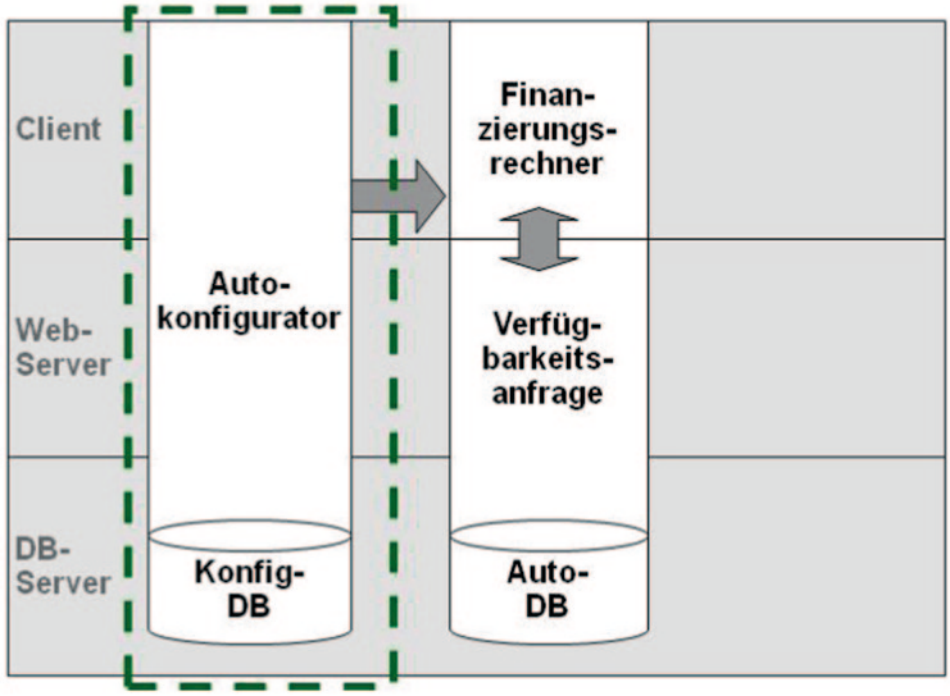


Abb. 9.5 Integration Autokonfigurator

ner mit dem Subsystem <Autokonfigurator> verbunden werden, das schon als einzeln aufzurufende Funktion produktiv im Einsatz ist. Die nächste Testphase beweist, dass der Autokonfigurator den Finanzierungsrechner fehlerfrei aufruft und den Listenpreis des konfigurierten Fahrzeugs korrekt als Kaufpreis übergibt. Das System besteht nun aus den in Abb. 9.5 gezeigten Komponenten.

3. **Integrationsstufe = Gesamtsystem:**

Gehen wir davon aus, dass auch die Vertragsabwicklung in mehreren Integrationsphasen zu einem Subsystem herangereift ist, so kann mit der letzten Integrationsstufe das gesamte System im Zusammenspiel getestet werden. (Der Übersichtlichkeit halber sind einige Komponenten wie Login-Prozedur, Firewall und Load-Balancer außen vor gelassen.)

Der Finanzierungsrechner tauscht mit der Vertragsabwicklung keine Daten auf direktem Wege aus, sondern die Vertragsabwicklung liest einmal täglich die benötigten Informationen aus dem Datenhaltungssystem des Finanzierungsrechners aus.[5] Daher sind für die bisher beschriebenen Integrationstests keine Treiber oder Platzhalter für diese indirekte und zeitversetzte Kommunikation notwendig gewesen. Wie Abb. 9.6 zeigt, hat das Subsystem <Vertragsabwicklung> das Gesamtsystem um einen Mail-

[5] Die Schnittstelle ist in Abb. 9.6 als schmaler Pfeil dargestellt.

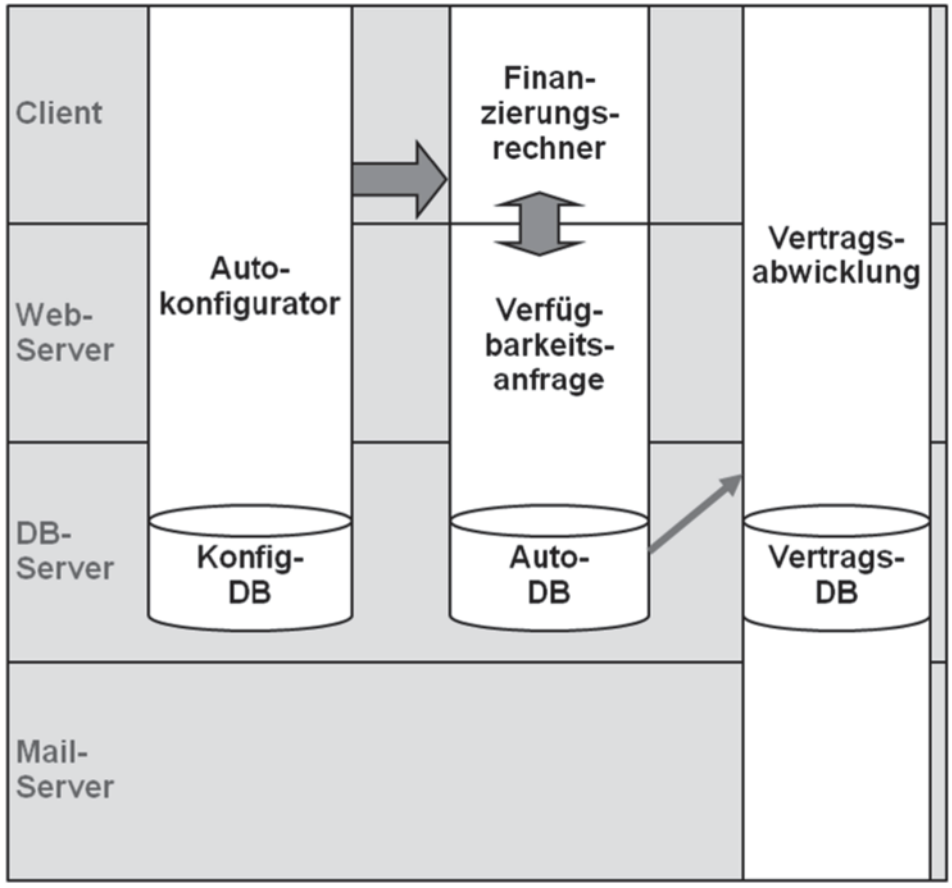


Abb. 9.6 Gesamtsystem

Server erweitert, weil damit verbindliche Finanzierungsangebote per E-Mail an den Kunden versendet werden können.

Das Anwendungssystem steht nun mit getesteten Schnittstellen für den anschließenden Systemtest zu Verfügung.

Dieses ausführliche Beispiel macht deutlich, wie sehr die einzelnen Schritte des Integrationstests von den zu integrierenden Komponenten und den verwendeten Webtechnologien abhängen. Für jede Integrationsstufe müssen fachliche und technische Testfälle entworfen werden.

9.3.2 Integration externer Komponenten

Der Integrationstest von Webanwendungen beschränkt sich nicht nur auf selbst entwickelte Komponenten. Auch in das System eingebundene Standardsoftware und für den Einsatz

der Webapplikation benötigte Komponenten müssen beim Integrationstest berücksichtigt werden. Komponenten können sowohl auf Client-Seite als auch auf Server-Seite notwendig sein. Das sind zum Beispiel für den Client:

- ActiveX[GL]-Controls
- Ausführbare EXE-Dateien
- Java-Archive[GL]
- Dynamic Link Libraries (DLLs)
- Plugins[6]
- Web-Archive[GL]

Auf der Server-Seite können Komponenten wie Firewall und Proxy-Server[GL] eingesetzt werden.

9.3.3 Werkzeuge für den Integrationstest

Bei der Planung der Tests muss berücksichtigt werden, dass je nach Integrationsstrategie **Testtreiber**, **Platzhalter** und/oder **Mocks** notwendig sind und programmiert werden müssen. Ggf. können sie schon im Komponententest eingesetzt und für den Integrationstest übernommen werden.

Der Testschwerpunkt liegt beim Integrationstest auf der Kommunikation der Schnittstellen. Daher ist es sehr hilfreich, den Datenverkehr auf den Schnittstellen aufzuzeichnen, sei es durch sogenannte **Monitore** (s. Kap. 12.9) oder eigens programmierte **Protokollausgaben**, die im produktiven Betrieb abgeschaltet werden können.

Wenn Komponenten der zu testenden Anwendung mit mobilen Endgeräten kommunizieren, werden spätestens für den Integrationstest **Simulatoren**[GL] und/oder **Emulatoren**[GL] benötigt. Die Aspekte und Tools des Mobile-Testens werden im Kap. 9.12 ausführlich beschrieben.

Für den Integrationstest können **Capture Replay Tools** zur Testautomatisierung (s. Kap. 14.3) eingesetzt werden. Oftmals können aber technische Ausnahmesituationen effizienter manuell als automatisiert getestet werden. Daher sollte der Einsatz von Capture Replay Tools im Integrationstest situationsabhängig erfolgen.

9.3.4 Qualitätsanforderungen zum Integrationstest

Die Qualitätsanforderung an den Integrationstest ist die 100-prozentige Erfüllung aller für die einzelnen Integrationsstufen definierten Testfälle, die zuvor von Fachleuten entworfen, geprüft und priorisiert wurden. Mit diesen nach Blackbox- und Whitebox-Verfahren

[6] Dem Test von Plugins ist das Kap. 9.7 gewidmet.

entworfenen Testfällen werden die funktionalen Anforderungen an die Schnittstellen überprüft.

9.3.5 Empfehlungen zum Integrationstest

Für die einzelnen Schritte des Integrationstests dürfen nur von der Qualitätssicherung freigegebene Komponenten in eine nächst höhere Integrationsteststufe übernommen werden, denn wenn beim Integrationstest Fehlerwirkungen auftreten, sollten diese eindeutig der Schnittstellenfunktionalität zugeordnet werden können.

Der Integrationstest ist ein typischer Greybox-Test, denn wie am Beispiel der oben beschriebenen SOAP-Kommunikation zu sehen ist, ist technisches Detailwissen für die Integration webbasierter Anwendungen notwendig. Daher sollten Spezialisten für die eingesetzten Technologien beim Integrationstest mitarbeiten und die systemtechnischen Testfälle entwerfen und priorisieren.

9.4 Funktionaler Systemtest[7]

Ist ein Subsystem fertig getestet, so wird es als eigenständige Komponente in einen übergeordneten Integrationstest übernommen. Dieses Verfahren wird so lange angewendet, bis in der letzten Stufe das Gesamtsystem zusammengeführt ist und für den funktionalen Systemtest in einer produktionsnahen Testumgebung zur Verfügung gestellt wird.

► Im **funktionalen Systemtest** das Gesamtsystem mit dem Ziel getestet, die korrekte und vollständige Umsetzung der funktionalen Anforderungen nachzuweisen.

9.4.1 Testszenarien zum funktionalen Systemtest

Der funktionale Systemtest überprüft – wenig überraschend – das Qualitätsmerkmal Funktionalität, allerdings mit Sicht auf fachliche Geschäftsprozesse. Dazu werden Testfälle und Testszenarien insbesondere nach den Methoden der zustands- und anwendungsfallbasierten Testfallermittlung (s. Abschn. 4.1.4 und 4.1.5) entworfen, mit denen die in der Anforderungsspezifikation beschriebenen Anwendungs- und Geschäftsvorfälle systematisch getestet werden. Diese Testszenarien zum Systemtest stellen sicher, dass alle Systemschnittstellen und alle Funktionen über das gesamte System hinweg korrekt arbeiten und den fachlichen Anforderungen entsprechen.

[7] Der funktionale Systemtest ist in diesem Kapitel als Testart beschrieben. Unter dem umfassenderen Begriff Systemtest wird eine Teststufe verstanden, (s. Kap. 16.1).

Beispiel eines Testszenarios im Systemtest

Das Testszenario, das den einfachsten Geschäftsvorfall der Finanzierungsanfrage abbildet und sich durch das gesamte Anwendungssystem bewegt, besteht aus folgenden Anwendungsfällen:

1. Der Kunde konfiguriert ein Auto mit dem Autokonfigurator.
2. Der Kunde berechnet eine Finanzierungsmöglichkeit im Finanzierungsrechner.
3. Der Kunde stellt zur Finanzierungsmöglichkeit eine Finanzierungsanfrage, die nach einer positiv beantworteten Verfügbarkeitsanfrage gespeichert wird.
4. Die Vertragsabwicklung erstellt anhand der Finanzierungsanfrage ein verbindliches Angebot und versendet es per E-Mail an den Kunden.

Für den funktionalen Systemtest müssen weitere Testszenarien entworfen werden, welche die komplexeren Geschäftsvorfälle und Abweichungen vom Standardprozess abbilden. Sie beinhalten zum Beispiel, dass mehrere Finanzierungsmöglichkeiten gespeichert werden und später eine davon für ein Angebot ausgewählt wird, dass eine Verfügbarkeitsanfrage nicht positiv beschieden wird und dass ein Angebot per Post versendet wird.

9.4.2 Werkzeuge und Qualitätsanforderungen zum funktionalen Systemtest

Zum funktionalen Systemtest kommen, bis auf die nicht mehr benötigten Treiber und Platzhalter, die gleichen Werkzeuge wie beim Integrationstest zum Einsatz, d. h. Capture Replay Tools, Simulatoren[GL] und Emulatoren[GL].

Zum funktionalen Systemtest wird der Nachweis eines festgelegten Testfallüberdeckungsgrades gefordert, wie zum Beispiel eine Testfallüberdeckung der Systemtestszenarien von 80 %.

Die beschriebenen Systemtestszenarien wiederum decken zu 100 % die spezifizierten Anwendungsfälle ab.

9.5 Link-Test

Hyperlinks[GL] (kurz Links) sind ein wichtiges Steuerungsinstrument von Webanwendungen. Websites beinhalten interne und externe Links, die nicht ins Leere laufen dürfen. Neben der Funktionalität der Links muss gewährleistet sein, dass keine unerlaubten Link-Typen benutzt werden.

► Der **Link-Test** überprüft die Rechtmäßigkeit und Fehlerfreiheit von internen und externen Links.

9.5.1 Link-Typen

Der Link-Test stellt sicher, dass jeder auf der Website angebrachte Link sein Ziel findet und dass keine „Gemeinheiten“ in der Website versteckt sind, denn Deep Links, Inline Links, Framing und Metatags mit Namen und Produkten von Mitbewerbern sind verboten.

Ein **Deep Link** verweist direkt auf eine fremde, untergeordnete Webseite. Dabei wird die Homepage der verlinkten Seite umgangen. Wird der Benutzer nicht darauf hingewiesen, entsteht für ihn der Eindruck, dass der angezeigte Inhalt zur verweisenden Website gehört. Ein Deep Link kann somit Urheberrechte verletzen, denn das Recht der körperlichen Verwertung (Vervielfältigung, Verbreitung, Ausstellung) liegt ausschließlich beim Verfasser.

Ein **Inline Link** stellt eine Grafik, die einen anderen Ursprung als die Website selbst hat, als Teil dieser Website dar. Wird der Benutzer nicht darauf hingewiesen, dass sich diese Grafik nicht auf dem gleichen Webserver wie die Website befindet, können die Urheberrechte verletzt werden.

Wie das Setzen von Inline Links ist auch das Framing ohne Erlaubnis des Betreibers der verlinkten Seite rechtlich unzulässig. Beim Framing wird eine fremde Webseite oder gar eine komplette Website in einem Frame der eigenen Website eingebettet (inline Frame, kurz **iFrame**).

Metatags enthalten Angaben im Quelltext eines HTML-Dokuments, die der Benutzer nicht sieht, die aber Webservern und Suchmaschinen Auskunft über die Website liefern. Solche Metatags dürfen nicht auf Namen und Produkte von Mitbewerbern verweisen, da sonst das Markenrecht verletzt wird. So könnten auf diesem indirekten Wege durch Suchmaschinen Benutzer irregeführt und auf Webseiten umgeleitet, sozusagen „gelinkt“ werden.

Mit diesen Erkenntnissen lässt sich eine Checkliste zum Link-Test zusammenstellen:

✓ Checkliste Link-Test

- Werden keine Deep Links verwendet?
- Werden keine Inline Links verwendet?
- Wird kein Framing eingesetzt?
- Finden alle internen Links Ihr Ziel?
- Werden interne Links im gleichen Fenster geöffnet?
- Ist jeder externe Link als solcher gekennzeichnet?
- Finden alle externen Links Ihr Ziel?
- Werden externe Links in einem neuen Fenster geöffnet?
- Funktionieren alle Links, die eine E-Mail generieren (Mail-to-Links)?
- Funktionieren alle Links, die ein Plugin aufrufen (z. B. einen Mediaplayer oder einen PDF-Reader)?
- Ist hinter jedem Link, der zu einem Dokument führt, der Dokumenttyp und die Dateigröße angegeben?

- Wenn eine Grafik mit einem Link versehen ist, ist der Link für den Benutzer erkenntlich – insbesondere, wenn die Darstellung von Grafiken in den Browser-Einstellungen unterbunden ist?
- Sind Grafiken mit mehreren Links (Imagemaps) erkennbar und sind die einzelnen Links eindeutig zu unterscheiden?
- Werden keine Namen und Produkte von fremden Unternehmen oder Mitbewerbern in Metatags verwendet und auch nicht in unsichtbarem Text?
- Können die AGB lokal gespeichert und gedruckt werden?

9.5.2 Werkzeuge für den Link-Test

Mit dem Einsatz von **Link Checkern** lässt sich die Funktionalität von Links ohne großen Aufwand feststellen. Im Internet gibt es zahlreiche kostenlose Möglichkeiten zur Überprüfung von Links.

Sehr bekannt sind der frei verfügbare **Link Checker des W3C** [URL: W3C-checklink] oder das einfach zu bedienende Freeware-**Tool Xenu's Link Sleuth**™ [URL: XenuLink].

Weitere Tools sind unter [URL: TestToolsSQA] in der Rubrik „Link Checkers" zu finden.

9.5.3 Qualitätsanforderungen zum Link-Test

Die Qualitätsanforderungen zum Link-Test bestehen aus dem Erfüllungsgrad der Checkliste „Link-Test" zu 100 %. Zum Nachweis müssen sowohl die rechtliche Unbedenklichkeit der Links und Metatags in einem Review-Protokoll als auch die Funktionalität der Links in einem – eventuell automatisch erstellten – Testprotokoll bestätigt werden.

9.5.4 Empfehlungen zum Link-Test

Link-Test mit Content-Test verbinden
Die Prüfung auf nicht erlaubte Link-Typen und verbotene Metatags hat starken Bezug zum Content-Test (s. Kap. 10.1), der die Rechtskonformität eines Web-Angebotes sicherstellt. Daher sollte die Checkliste „Link-Test" schon beim Content-Test zu Rate gezogen werden. Der abschließende Link-Test kann allerdings erst nach der Fertigstellung der Website erfolgen.

Link-Test im laufenden Betrieb
Die Überprüfung der Links muss besonders in Hinsicht auf externe Links auch während des laufenden Betriebes regelmäßig erfolgen. Diese Tätigkeit kann ohne großen Aufwand mit Link Checkern automatisiert werden.

9.6 Cookie-Test

Cookies[GL] erlauben einem Webserver mit Hilfe des Browsers Informationen auf dem Rechner des Nutzers in einer Textdatei abzulegen. Das kann zum Beispiel bei einem Online-Shop notwendig sein, der wissen muss, welche Artikel ein Benutzer bereits in den Warenkorb gelegt hat.

► Der **Cookie-Test** überprüft die korrekte fachliche und technische Funktionalität von eingesetzten Cookies.

9.6.1 Überprüfung von Cookies

Die Ausführung einer Funktion, die Cookies benötigt, darf durch unerwartete Einstellungen des Browsers nicht verhindert werden. Sind zum Beispiel Cookies auf dem Client gesperrt, muss der Benutzer darauf hingewiesen werden.

Das kann dadurch geschehen, dass der Benutzer erst im Fehlerfall die Nachricht erhält, dass seine Browser-Einstellungen das Anlegen von Cookies nicht zulassen. Eine andere Möglichkeit ist die aktive Ansprache des Benutzers, indem er mit dem Aufruf der entsprechenden Funktion einen Hinweis bekommt, dass Cookies benötigt werden und ob seine Browser-Einstellungen den Einsatz von Cookies zulassen. Diese zweite Lösung hat den Vorteil, dass der Benutzer auf jeden Fall die Information erhält, dass Cookies eingesetzt werden und ggf. Maßnahmen ergreifen kann.

Der Cookie-Test prüft also nicht nur die fachliche und technische Funktionalität eines Cookies, sondern auch die Erfüllung der Informationspflicht gegenüber dem Benutzer.

✓ Checkliste Cookie-Test

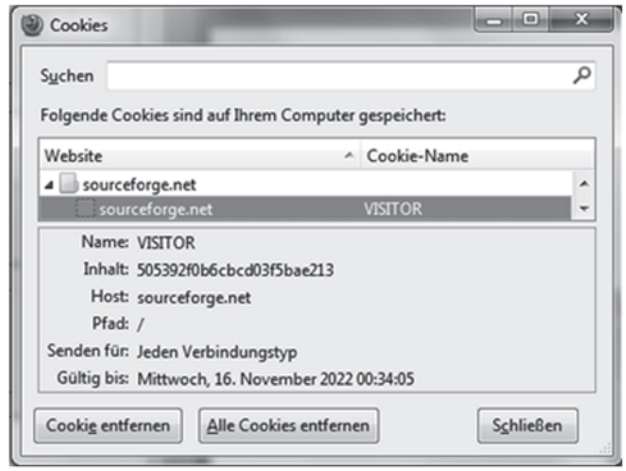
Funktion

- Werden die Inhalte des Cookies von der Anwendung korrekt gespeichert und wieder ausgelesen?
- Wird das Cookie nach der Sitzung korrekt gelöscht oder besitzt das Cookie ein Verfallsdatum?
- Arbeitet die Anwendung korrekt, wenn der Nutzer das Cookie löscht, insbesondere während der Sitzung?
- Funktioniert das Cookie, wenn die Sitzung unterbrochen und wieder aufgenommen wird?
- Wird der Benutzer bei dynamischer IP-Adresse richtig identifiziert?

Unterrichtung

- Wird der Benutzer darauf hingewiesen, wenn er in seinen Browser-Einstellungen Cookies unterbunden hat?
- Wird der Benutzer unterrichtet, wenn Cookies verwendet werden?

Abb. 9.7 Cookie



- Wird darauf hingewiesen, wie die Cookies verwendet werden?
- Findet die Unterrichtung statt, bevor die Cookies gesetzt werden?
- Muss der Benutzer dem Einsatz von Cookies explizit zustimmen und wird auch kein Cookie gesetzt, wenn er ablehnt?
- Ist festgelegt, wie lange die Website verlassen werden darf, ohne dass Informationen verloren gehen, und wird der Benutzer ggf. beim Verlassen der Website darüber informiert?

Zu jedem Prüfpunkt dieser Checkliste müssen Testfälle beschrieben und getestet werden.

9.6.2 Werkzeuge für den Cookie-Test

Unter den Stichworten „**Cookie Manager**“ oder „**Cookie Viewer**“ sind im Internet Werkzeuge zu finden, mit denen die Inhalte von Cookies überprüft, geändert und die Cookies gelöscht werden können.

Diese Möglichkeiten bieten auch einige Browser. Zum Beispiel können mit der Funktion „Cookies anzeigen“ des Browsers Firefox unter anderem Herkunft, Inhalt und Gültigkeitsdatum eines ausgewählten Cookies angezeigt werden (s. Abb. 9.7).

9.6.3 Qualitätsanforderungen zum Cookie-Test

Die Qualitätsanforderung an den Cookie-Test ist der Nachweis, dass alle zum Cookie-Test definierten Testfälle erfolgreich getestet wurden (100 % Testfallüberdeckung).

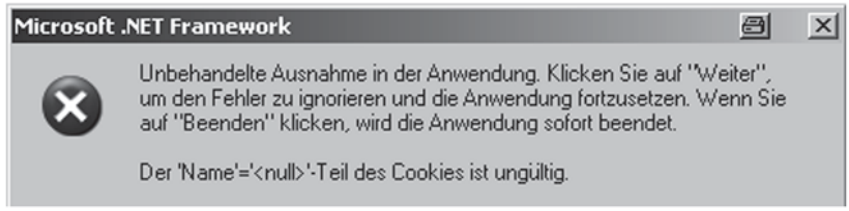


Abb. 9.8 Fehler Cookie-Test

Bei der Beschreibung der Testfälle ist darauf zu achten, dass sie alle Punkte der obigen Checkliste abdecken. Neben diesen Standardtestfällen müssen auch die anwendungsspezifischen Anforderungen an die Cookies durch fachliche Testfälle abgedeckt werden.

Eine ganz spezielle Qualitätsanforderung ist der **Verzicht auf Cookies**. Cookies sind aus Datenschutzgründen kritisch und oft in den Browser-Einstellungen deaktiviert, sei es freiwillig durch den Benutzer oder par ordre du mufti. Im Cookie-Test muss das Einhalten dieser Qualitätsanforderung, d. h. das Nicht-Verwenden, geprüft werden.

9.6.4 Empfehlungen zum Cookie-Test

Cookies werden von einzelnen Komponenten einer Website gesetzt und gelesen, daher sollten Cookie-Tests in der Testphase Komponententest durchgeführt werden. Cookies werden in der Regel nicht spezifiziert, was bedeutet, dass der Tester gefordert ist und technisch orientierte Testfälle designen muss.

Ein Beispiel zum Cookie-Test

Der Entwickler: Eine zu testende.NET-Anwendung speichert auf Wunsch beim Login das Passwort in einem Cookie und merkt sich – wo auch immer –, dass sie das Passwort im Cookie abgelegt hat. Der Entwickler hat sich die schöne Funktion ausgedacht, dass der User aus der Anwendung heraus das Cookie löschen kann.

Der Tester: Das Cookie wird korrekt gelöscht. Nun will sich der User, sprich Tester, wieder anmelden. Der Login-Prozess weiß, dass er das Passwort im Cookie gespeichert hat und bietet das Login-Fenster gar nicht mehr an, er holt das Passwort aus dem Cookie; zumindest versucht er es… s. Abb. 9.8.

Der verzweifelte Tester: Und, ganz große Wirkung: Weder die Deinstallation und erneute Installation der Anwendung inkl. des gesamten.NET-Frameworks behebt dieses Problem.

Der entnervte Entwickler: Nach aufwändigen Untersuchungen und Internetrecherchen: Erst das Löschen tief vergrabener Registry-Einträge erlaubte die Neuinstallation der fehlerbereinigten Version der Anwendung.

9.7 Plugin-Test

In der Regel hat jeder Internet-Benutzer Plugins installiert, die zusätzlich zum Browser eingesetzt werden. Ein **Plugin** ist ein Programmmodul, das in eine bestehende Anwendung integriert werden kann und deren Funktionsumfang erweitert. Es ist ohne die dafür vorgesehene Anwendung nicht funktionsfähig. Ein Plugin ist vom verwendeten Betriebssystem abhängig und muss separat von der Anwendung auf dem Client installiert werden.

Ein bekanntes Beispiel ist das Adobe Reader Plugin, das für verschiedene Browser installiert werden kann. Es wird benötigt, um Dokumente lesen zu können, die im PDF[GL]-Format zur Verfügung gestellt werden.

Rich Internet Applications[GL] (RIA), die u. a. in einem Webbrowser laufen, erlauben „reichhaltige“ Darstellungs- und besondere Interaktionsmöglichkeiten für den Benutzer. Beispiele für RIA-Techniken sind Browser-Spiele oder der Webshop, in dem Visitenkarten, Kalender und T-Shirts selbst gestaltet werden können.

Zu pluginbasierten RIA-Technologien gehören u. a. Flex[GL] (ein Apache-Projekt), Silverlight[GL] von Microsoft® und JavaFX[GL] von Oracle®.

► Der **Plugin-Test** überprüft eine Webanwendung auf fehlerfreies Verhalten und korrekte Benutzerführung in Bezug auf benötigte Plugins.

9.7.1 Szenarien zum Plugin-Test

Beim Plugin-Test werden zwei generelle Systemsituationen geprüft. Zum einen kann ein Plugin auf dem Benutzer-Client fehlen und zum anderen kann ein Plugin zu viel installiert sein.

Falls ein von einer Anwendungskomponente benötigtes Plugin nicht auf dem Client installiert ist, muss der Benutzer einen entsprechenden Hinweis erhalten und auf der Website die Möglichkeit erhalten, das Plugin aus dem Internet zu laden.

Falls auf dem Rechner des Benutzers eine Software-Komponente installiert ist, die mit der neuen Webanwendung unverträglich ist, muss diese Situation automatisch erkannt werden und vom Benutzer behoben werden können. Sonst kann es passieren, dass die Anwendung nicht oder nicht korrekt funktioniert, wie die folgende Testgeschichte eines Plugins belegt.

Szen(ari)en eines Plugin-Tests, eine wahre Geschichte

Eine Webapplikation ist in Java realisiert und benötigt für die Ausführung von Java-Applets[GL], die mit Swing[GL]-Klassen arbeiten, das aktuelle Java-Plugin in der Version 1.4.1.

Eigentlich soll nur die Funktion einer Webanwendung getestet werden, die zu ihrer Ausführung das besagte Java-Plugin benötigt. Auf dem Rechner des Testers ist – was dieser aber noch nicht weiß – die ältere Version 1.3 des benötigten Plugins installiert.

Szenario Plugin-Test – Teil 1

Der Test beginnt. Eine wichtige Funktion der Webanwendung kann nicht aufgerufen werden. Es stellt sich heraus, dass die Anwendung nicht die Existenz der Version 1.4.1 des benötigten Plugins auf dem Client-PC überprüft, sondern nur das Vorhandensein irgendeines Java-Plugins dieses Herstellers. Die ältere Version 1.3 auf dem Testrechner, auf welche die Webanwendung zugreift, lässt aber keine Swing-Klassen zu.

Szenario Plugin-Test – Teil 2

Der Fehler wird behoben, d. h. beim Aufruf der Webanwendung wird nun geprüft, ob die richtige Version des benötigten Plugins installiert ist. Ist das nicht der Fall, wird der Benutzer darauf hingewiesen, dass er ein neues Plugin mindestens in der Version 1.4.1 installieren muss. Es wird auch der richtige Download-Link im Rahmen des Hinweises zur Verfügung gestellt. Die Version 1.3.1 wurde inzwischen vom Testrechner deinstalliert. So weit, so gut.

Szenario Plugin-Test – Teil 3

Der anschließende Test deckt das nächste Problem auf: Das Plugin kann nicht installiert werden, weil der Tester sich nicht mit Administratorrechten auf seinem Rechner angemeldet hat. Da der verzweifelte Benutzer nicht auf seine fehlenden Rechte hingewiesen wird, weiß er nicht, warum sich das Plugin nicht installieren lässt. Das ist zwar kein Fehler der Webapplikation, sondern ein Versäumnis des Plugin-Herstellers, aber die Wirkung ist fatal.

Szenario Plugin-Test – Teil 4

Nachdem dieses Problem erkannt ist und der Benutzer einen Hinweis zu den benötigten Rechten erhält, meldet sich der Tester mit Administratorrechten an seinem Rechner an. Das neue Plugin wird erfolgreich installiert.

Die Anwendung funktioniert aber trotzdem nicht richtig. Der Grund liegt im Betriebssystem. Das greift nun auf ein zweites Plugin eines anderen Software-Herstellers zu, das bereits früher auf dem Rechner installiert wurde.

Diese Situation tritt allerdings erst auf, seitdem die Version 1.3.1 deinstalliert worden ist – versteht sich. Erst nachdem das unverträgliche Plugin in den Systemeinstellungen deaktiviert ist, kann die eigentliche Funktion getestet werden.

Wie die Geschichte[8] zeigt, ist es für den Plugin-Test wichtig zu wissen, welche Plugins von Benutzern oft eingesetzt werden, welche Plugins ein Browser-Typ unter welchem Betriebssystem benötigt und welche Unverträglichkeiten mit der zu testenden Webanwendung auftreten können.

Die oben beschriebenen Situationen führen zur Checkliste „Plugin-Test“.

[8] Die kleine Testgeschichte zum Java-Plugin ist noch nicht zu Ende. Sie wird im Abschn. 14.2.3, Regressionstest, fortgesetzt.

✓ **Checkliste Plugin-Test**

- Ist auf der Website angegeben, für welche Plugins in welcher Version die Fehlerfreiheit der Anwendung garantiert wird?
- Hat der Benutzer die Möglichkeit, die benötigten Plugins direkt von der Website aus dem Internet zu laden?
- Sind die Installationsprozesse der benötigten Plugins getestet?
- Wird der Benutzer auf die Rechte hingewiesen, die er zur Installation eines Plugins benötigt?
- Liegen alle notwendigen Zertifikate vor und können sie installiert werden (zum Beispiel für „Cut & Paste"-Funktionen aus einem Java-Applet heraus)?
- Ist die Funktionalität der benötigten Plugins in Kombination mit möglicherweise bereits installierten, alten Plugin-Versionen des Herstellers getestet?
- Ist die Funktionalität der benötigten Plugins in Kombination mit möglicherweise bereits installierten, neueren Plugin-Versionen des Herstellers getestet?
- Ist die Funktionalität der benötigten Plugins in Kombination mit möglicherweise bereits installierten Plugins anderer Hersteller sichergestellt?
- Sind mögliche Unverträglichkeiten mit anderen Software-Komponenten (keine Plugins) geprüft und Gegenmaßnahmen beschrieben?

Ein Beispiel für Abhängigkeitsphänomene

Zum letzten Punkt der Checkliste ein Beispiel aus einem Projekt. Ein gängiger PDF-Reader konnte nicht geöffnet werden, solange eine bestimmte DLL auf dem Client-PC installiert war. Diese DLL wurde von einem Programm benötigt, das von der zu testenden Webanwendung unabhängig war – eigentlich.

Die Ursache lag in der eingesetzten Betriebssystemversion und konnte erklärt, aber nicht behoben werden.

9.7.2 Werkzeuge für den Plugin-Test

Für die Vorbereitung und Planung der Plugin-Tests können **User Tracking Tools** (s. Abschn. 10.3.6) wichtige Informationen liefern. Sie erstellen zum Beispiel Übersichten, die zeigen, welche Plugins am häufigsten von den Besuchern einer Website verwendet werden.

Die Testszenarien zum Plugin-Test können mit **Capture Replay Tools** (s. Kap. 14.3) automatisiert werden. Allerdings sollte für solche technischen Tests, die auch einfach manuell durchgeführt werden können und relativ selten wiederholt werden müssen, im konkreten Fall der Nutzen einer Automatisierung hinterfragt werden.

Auch oder gerade Plugins, die auf RIA-Technologien basieren, müssen mit **Load Test Tools** (s. Kap. 12.9) auf Performanz getestet werden, die dann natürlich die entsprechende RIA-Technologie „bedienen" können müssen.

9.7.3 Qualitätsanforderungen zum Plugin-Test

Das Hauptaugenmerk des Plugin-Tests liegt auf der Interoperabilität (s. Kap. 9.11), die ein Teilmerkmal des Qualitätsmerkmals Funktionalität ist. Die Interoperabilität eines zu testenden Plugins wird durch die 100-prozentige Überdeckung der Testfälle nachgewiesen, die auf Basis der obigen Checkliste entworfen werden.

9.7.4 Empfehlungen zum Plugin-Test

Der Plugin-Test wird schon mit dem funktionalen Komponententest begonnen, sofern Komponenten der Anwendung Plugins aktiv nutzen. Während des laufenden Betriebes des Systems muss der Plugin-Test mit überarbeiteten Testszenarien immer dann wiederholt werden, wenn sich die Systemlandschaft verändert.[9]

Beim Testen von und mit Plugins müssen in den betreffenden Testumgebungen automatische Updates abgeschaltet werden, damit sich die Testumgebungen nicht unkontrolliert ändern.

Der Plugin-Test sollte von Testern mit entsprechendem technischem Wissen durchgeführt werden. Der Test ist sehr wichtig, denn wenn das komplexe, technische Zusammenspiel von Software-Komponenten, Browsern und Betriebssystemen nicht funktioniert, ist im schlimmsten Fall die gesamte Webanwendung unbrauchbar.

Informationen zu Java-Plugins stehen unter [URL: Plugin-Java]. Aktuelle Listen der verfügbaren und am häufigsten eingesetzten Plugins für Mozilla-Produkte wie den Firefox-Browser sind in [URL: PluginDoc] zu finden.

9.8 Test der Browser-Einstellungen

Eine Erfahrung beim Online-Banking

Meine Hausbank wollte „nur“ den Aufruf des bislang einwandfrei funktionierenden Programms zum Online-Banking ändern. Ab dem angekündigten Umstellungstermin war aber kein Online-Banking mehr möglich. Meine Vermutung war, dass der besagte Termin verschoben worden war.

Dem war aber nicht so. Nach ein paar Tagen und eher zufällig stieß ich darauf, dass für die neue Online-Banking-Funktion PopUp-Windows zugelassen sein müssen. Diese wurden aber aufgrund meiner Browser-Einstellungen für die Website der Bank unterdrückt. Leider hatte die Anwendung für diesen Fall keinen Hinweis für den Benutzer vorgesehen.

[9] …wie das im Abschn. 14.2.3 fortgeführte Beispiel zum Plugin-Test unterstreicht.

Unerwünschte Browser-Einstellungen können eine Webapplikation bis hin zur Funktionsuntüchtigkeit beeinflussen. Daher ist unter Einbeziehung der Risikoklassen (s. Tab. 9.5) die Reaktion der Anwendung auf unterschiedliche Browser-Einstellungen zu testen.

9.8.1 Testfälle für die Browser-Einstellungen

Die Testfälle, die für den Test der Browser- Einstellungen entworfen werden müssen, werden aus der Checkliste „Browser-Einstellungen“ abgeleitet.

✓ Checkliste Browser-Einstellungen

- Arbeitet und erscheint die Anwendung korrekt, wenn in den Browser-Einstellungen ein Element nicht oder nur beschränkt erlaubt ist?
- Wird dem Benutzer ein entsprechender Hinweis gegeben, wenn ein Element benötigt wird, aber nicht verfügbar ist?
- Wurden die Funktionen und Reaktionen der Anwendung mit den folgenden Einstellungen überprüft?
  - Cookies[GL] sind aktiviert/deaktiviert
  - Cookies werden nur von der Original-Website akzeptiert
  - Das Laden von Grafiken ist aktiviert/deaktiviert
  - Nur Grafiken vom Ursprungs-Server werden geladen
  - Animierte Grafiken werden nicht wiederholt
  - Animierte Grafiken werden nur einmal wiederholt
  - PopUp-Fenster werden blockiert/nicht blockiert
  - ActiveX[GL] Steuerelemente sind aktiviert/deaktiviert
  - NET-Komponenten[GL] sind aktiviert/deaktiviert
  - Java ist aktiviert/deaktiviert
  - JavaScript ist aktiviert/deaktiviert
  - Weitere mögliche JavaScript-Einstellungen sind jeweils aktiviert/deaktiviert

9.8.2 Werkzeuge zum Test der Browser-Einstellungen

Einige Browser-Einstellungen können recht einfach mit der **Web Accessibility Toolbar** (**WAT**, [URL: AbITools]) überprüft werden. Das Werkzeug ist für den Internet Explorer und für Opera verfügbar und für den nicht kommerziellen Gebrauch kostenlos.

Ein ähnliches Tool für Firefox ist die kostenlose **Web Developer Toolbar** (**WDT**, [URL: WebDev]), einem Firefox Add-on (s. Abb. 9.9).

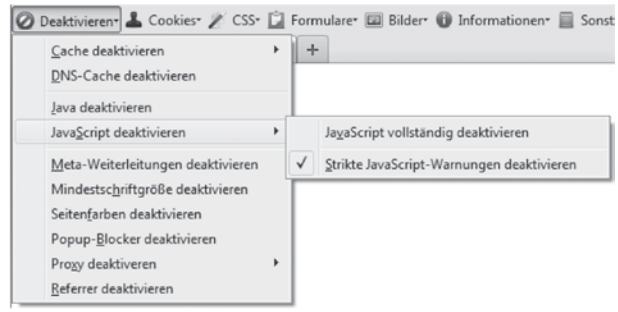


Abb. 9.9 Web Developer Toolbar

9.9 Webservice-Test

Ein **Webservice** ist eine Software-Anwendung, die im Netz eine Dienstleistung bereitstellt.

Aus technischer Sicht automatisiert ein Webservice unabhängig von Plattformen und Programmiersprachen die XML-basierte Kommunikation zwischen Software-Systemen über lokale oder globale Netzwerke, insbesondere über das Internet. Ein Webservice ist durch einen Uniform Resource Identifier (URI) eindeutig identifizierbar, seine Schnittstellen sind als XML-Artefakte definiert, beschrieben und auffindbar.

9.9.1 Webservice aus Testersicht

Aus Testersicht ist ein Webservice eine eigenständige Komponente, d. h. wohl definiert und separat ausführbar, und muss daher einem Komponententest unterzogen werden. Auf Basis der Beschreibung des Webservices werden für den Funktionstest fachliche Testfälle nach den klassischen Testfallentwurfsverfahren erstellt und ausgeführt.

Dummerweise hat ein Webservice keine eigene Benutzeroberfläche, über die Testfälle im Komponententest ausgeführt werden könnten. Daher werden für den Aufruf von Webservices, zumindest im funktionalen Komponententest, Testtreiber benötigt.

► Der **Webservice-Test** ist ein funktionaler Komponententest zu einem Webservice, der mit geeigneten Testtreibern ausgeführt wird.

9.9.2 Werkzeuge für den Webservice-Test

Für den Test von Webservices bieten sich als Testtreiber spezialisierte Open Source Tools wie **SoapUI** [URL: SoapUI] an. Aber auch Lasttesttools wie **HP LoadRunner** oder das

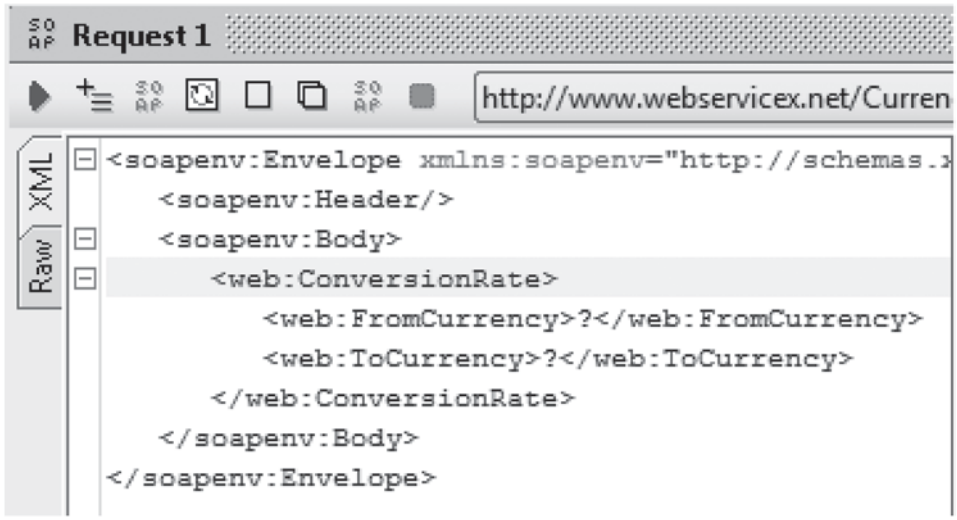


Abb. 9.10 WSDL in SoapUI

Open Source Tool **JMeter** [URL: JMeter] können als Testtreiber für Webservices eingesetzt werden.

Von SoapUI gibt es eine freie Version und eine Pro-Version, wobei die freie Version schon sehr ergiebig für das Testen von Webservices ist. In SoapUI wird die WSDL-Datei (Web Services Description Language)[GL], die den Webservices detailliert festlegt, eingeben und vom Tool ausgelesen.

Ein Beispiel zum Webservice-Test mit SoapUI

Unser Autokonfigurator soll inklusive Finanzierungsrechner in Russland eingeführt werden. Dafür sollen die Finanzierungsbeträge auch in Rubel umgerechnet werden können. Diese Funktionalität programmieren wir nicht neu, sondern greifen auf einen existierenden Webservice zu. Zum Beispiel wird auf www.webservicex.net der Währungsrechner „Currency Converter“ als Webservice mit seiner WSDL www.webservicex.net/CurrencyConvertor.asmx?WSDL zur Verfügung gestellt.

SoapUI liest die WSDL aus und liefert die beiden Eingabeparameter <FromCurrency> und <ToCurrency> des Webservices (s. Abb. 9.10). Anschließend lassen wir den Rubel rollen und Euro (EUR) in Rubel (RUB) umrechnen (s. Abb. 9.11).

Recht einfach lässt sich aus diesem einen erfolgreichen Request eine Testsuite aus mehreren Testfällen zusammenkopieren. Je Testfall können Bedingungen und konkret erwartete Ergebnisse festgelegt werden. Die Testergebnisse werden übersichtlich angezeigt (s. Abb. 9.12).

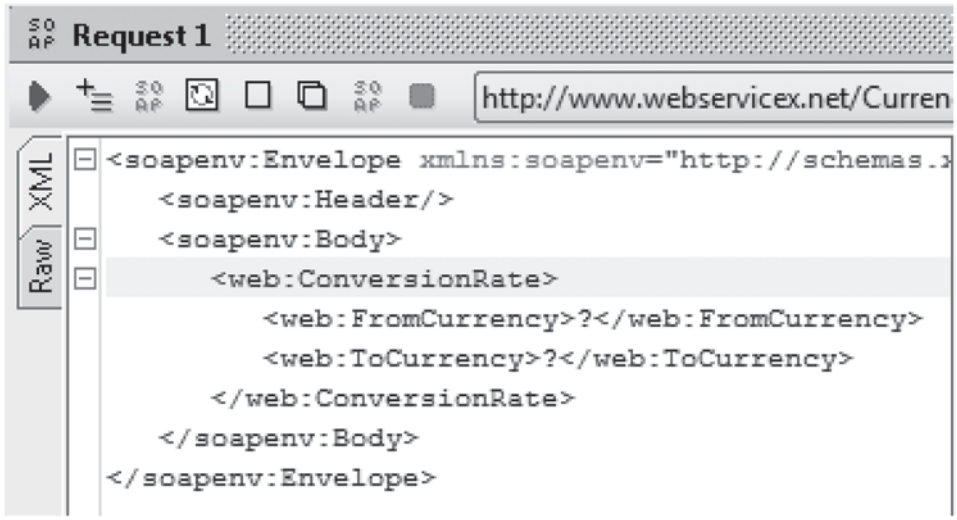


Abb. 9.11 Webservice in SoapUI

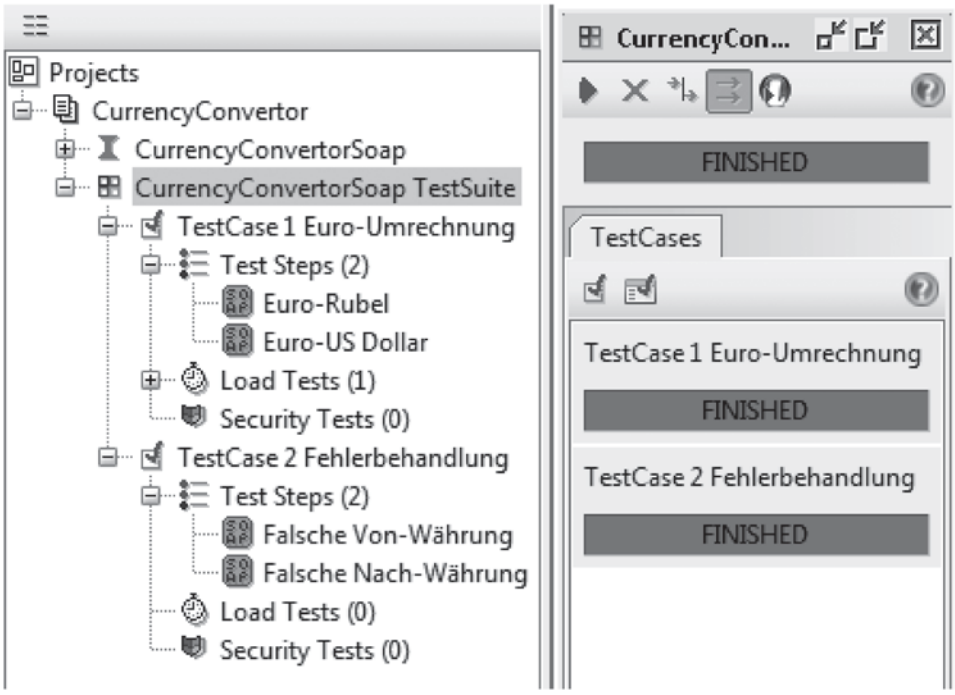


Abb. 9.12 Testsuites in SoapUI

9.9.3 Qualitätsanforderungen zum Webservice-Test

Die Qualitätsanforderungen zum Webservice-Test entsprechen denen des Komponententests, d. h. es müssen systematisch Testfälle beschrieben und mit der vorgegebenen Testfallüberdeckung getestet werden.

9.10 Sicherheitstest

Sicherheit (Security) einer Software ist als Teilmerkmal des Qualitätsmerkmales Funktionalität die Fähigkeit, nicht autorisierte Zugriffe auf Programme und Daten, die vorsätzlich oder versehentlich erfolgen, zu verhindern.

Je nach Aufgabenstellung, die eine Anwendung zu erfüllen hat, werden unterschiedliche Sicherheitsanforderungen an sie gestellt. Reine Informationsangebote müssen vor Manipulationen der angebotenen Inhalte sicher sein. Anwendungen, die sensible Daten verarbeiten, müssen vor fremden Zugriffen geschützt werden. Geschäftliche Transaktionen[GL] unterliegen noch schärferen Sicherheitsvorkehrungen, um Missbrauch zu verhindern.

► Der **Sicherheitstest** prüft Eignung, Korrektheit, Unumgänglichkeit und Wirksamkeit der in einem System eingesetzten Sicherheitsvorkehrungen.

Zu den **Sicherheitsvorkehrungen** eines Anwendungssystems zählen zum einen die speziell für die Anwendung programmierten Sicherheitsfunktionen und zum anderen die systemtechnischen Sicherheitsmechanismen.

9.10.1 Schritte des Sicherheitstests

Der Sicherheitstest geht in drei Schritten vor:

1. Eignung der Sicherheitsvorkehrungen überprüfen
2. Systemschwachstellen identifizieren
3. Penetrationstest durchführen

Schritt 1: Eignung der Sicherheitsvorkehrungen überprüfen
Die fachlichen und technischen Sicherheitsanforderungen an das System müssen zu Beginn des Sicherheitstests vollständig definiert sein. Sicherheitsexperten prüfen die Sicherheitsanforderungen in einem Review und stellen fest, ob die notwendigen Sicherheitsfunktionen und Sicherheitsmechanismen bereitgestellt werden und ob sie angemessen sind.

Schritt 2: Systemschwachstellen identifizieren
Wenn die Sicherheitsvorkehrungen als geeignet eingestuft worden sind, werden anschließend potentielle Schwachstellen des Systems identifiziert, die ermöglichen könnten, dass die Sicherheitsvorkehrungen umgangen oder außer Kraft gesetzt werden.

Schritt 3: Penetrationstest durchführen
Der Tester schlüpft beim Penetrationstest in die Rolle eines potentiellen Angreifers und versucht mit dessen Mitteln die Sicherheitsvorkehrungen zu umgehen. Das geschieht zum einen durch Manipulationen der Anwendung und zum anderen durch gezielte Angriffe auf den Webserver.

► Der **Penetrationstest** beweist oder widerlegt die Existenz der zuvor identifizierten Sicherheitsschwachstellen, indem gezielte Angriffe auf das Anwendungssystem simuliert werden.

Der Tester wird zum Beispiel versuchen,

- in der Webapplikation unerwartete Eingabekombinationen zu erzeugen,
- Reihenfolgen von Funktionen zu ändern,
- nicht erlaubte Funktionen auszuführen,
- Funktionen und Funktionsabfolgen zielgerichtet zu unterbrechen,
- an die Session-ID eines Benutzers zu gelangen, um dann mit dessen Rechten auf die Webanwendung zugreifen zu können (**Session-Hijacking**),
- den Passwortschutz zu umgehen und
- Passwörter durch Ausprobieren aller Möglichkeiten zu ermitteln (**Brute Force Angriff**)[10].

Angriffe auf den Webserver werden durchgeführt, um festzustellen, ob sie von den Sicherheitsvorkehrungen erkannt und abgewehrt werden:

- Mit einem automatisierten **Denial of Service (DoS) Angriff** wird versucht, durch eine hohe Anzahl von Verbindungsanfragen dem Webserver die Ressourcen zu entziehen und ihn für reguläre Anfragen zu blockieren. Wird der DoS-Angriff von vielen Computern gleichzeitig durchgeführt, spricht man von einem **Distributed Denial of Service** (DDoS) Angriff.
- Mit einer **SQL-Injektion** wird versucht, die Kontrolle über die Datenbank oder den Datenbank-Server zu erhalten. Dazu werden die Parameter einer Datenbankanfrage mit SQL-Steuerzeichen versehen, mit denen Datenbankbefehle in das System geschleust werden. Auf diese Weise können nicht zugängliche Daten ausgelesen oder verändert werden.
- Beim **Man in the Middle Angriff** (MitM) wird versucht, die Verbindung einer gesicherten Verschlüsselung auszuspionieren. Dazu schaltet sich der Übeltäter in die Kommunikation zwischen Benutzer und Webserver und manipuliert die ausgetauschten Informationen.

[10] Ein Beispiel für das „Knacken“ von Passwörtern ist bei heise.de zu finden: http://www.heise.de/-270518.html.

9.10.2 Planung und Durchführung des Sicherheitstests

Um Hackern möglichst wenig Angriffsfläche zu geben, dürfen für Benutzer und Suchmaschinen keine Informationen über das System (z. B. Hardware- und Softwareversionen) sichtbar oder erforschbar sein.

Zur Planung und Durchführung des Sicherheitstests dient die folgende Checkliste mit Fragen, die bei der Schwachstellenanalyse berücksichtigt und mit Testfällen überprüft werden müssen:

✓ Checkliste Sicherheitstest

Sicherheitsanforderungen

- Sind die Anforderungen an die systemtechnische Sicherheit festgelegt?
- Ist die systemtechnische Sicherheit vertraglich geregelt?
- Sind die Anforderungen an die anwendungsspezifischen Sicherheitsfunktionen festgelegt?
- Sind die Sicherheitsanforderungen vollständig beschrieben?
- Sind die Sicherheitsanforderungen in den Geschäftsprozessen berücksichtigt?
- Wurden die Sicherheitsanforderungen nachweisbar von Fachleuten geprüft?
- Liegt eine Schwachstellenanalyse vor, in der fachliche und technische Sicherheitsaspekte betrachtet werden?
- Wurden die erkannten Schwachstellen bewertet und bei Bedarf beseitigt?
- Wurde die Schwachstellenanalyse nachweisbar von Fachleuten geprüft?
- Sind die Sicherheitsvorkehrungen für den laufenden Betrieb eingeplant?

Hardware/Sicherheits-Software

- Werden korrekt arbeitende Proxy-Server[GL] eingesetzt?
- Meldet der Relay-Host[GL] Sicherheitsverletzungen korrekt an den Administrator?
- Ist vor dem Relay-Host nur unbedingt notwendige Software installiert (zum Beispiel auf Web- und FTP-Server)?
- Sind nur unbedingt notwendige Daemons[GL] installiert?
- Sind die neuesten Versionen der Daemons installiert?
- Wird X-Windows[GL]-Verkehr verhindert?
- Sind alle bekannten Security-Patches installiert?
- Werden Checksummenverfahren angewendet?
- Kann die Firewall nicht von innen umgangen werden?

Verschlüsselung

- Werden bei Transaktionen Verschlüsselungsverfahren eingesetzt?
- Werden gängige Verschlüsselungsverfahren verwendet?
- Werden die zur Ver- und Entschlüsselung genutzten Schlüssel passwortgeschützt oder verschlüsselt gespeichert (Überschlüsselung)?
- Ist der Überschlüssel geschützt?

Passwortschutz
- Sind Mitgliederbereiche passwortgeschützt?
- Haben Passwörter mindestens eine Länge von acht Zeichen?
- Werden Sonderzeichen und Ziffern in den Passwörtern erzwungen?
- Werden Passwortdateien und -felder verschlüsselt gespeichert?
- Können alte Passwörter nicht mehr aktiviert werden?
- Wird der Zugang nach mehreren falschen Passworteingaben gesperrt?
- Kann der Passwortschutz nicht deaktiviert werden?
- Kann die Passwortprüfroutine nicht unterbrochen werden?
- Kann die Passworteingabe nicht umgangen werden?
- Wird eine regelmäßige Änderung des Passwortes beim Benutzer eingefordert?

Prüfungen Benutzerführung
- Werden falsche Eingaben sinnvoll abgefangen?
- Werden sicherheitsrelevante Eingaben client- und serverseitig geprüft?
- Bleiben bei unerwarteter Browser-Navigation keine Transaktionen offen?
- Können Transaktionen nach einem Systemfehler nicht wieder aufgenommen werden?
- Hat der Benutzer die Möglichkeit, sich abzumelden?

Session Handling
- Sind Session-Ids eindeutig, zufällig generiert und für den Benutzer nicht sichtbar?
- Ist eine Session eindeutig einem Benutzer zugeordnet?
- Beendet ein Logout alle Sessions?
- Bleiben nach Beendigung der Session keine sensiblen Daten im Cache?
- Wird die Session automatisch beendet, wenn in einem vorgegebenen Zeitraum keine Eingabe durch den Benutzer erfolgt?
- Wird der Benutzer über den Zeitraum der automatischen Beendigung der Session informiert?

Geheimhaltung technischer Informationen
- Werden dem Benutzer keine Namen und Versionen von Systemkomponenten (Serverbetriebssystem, Datenbank, Anwendung,…) angezeigt?
- Namen und Versionen von Systemkomponenten (Serverbetriebssystem, Datenbank, Anwendung,…) können von Suchmaschinen nicht ausgelesen werden?
- Enthalten Fehlermeldungen des Systems an den Benutzer keine systemtechnischen Details?
- Werden keine Modellinformationen von Webkameras ausgegeben?

Protokollierung
- Werden Protokolldateien geschrieben?
- Enthalten die Protokolldateien keine sensiblen Daten?

- Werden die zu protokollierenden Aktivitäten (Anmeldungen, Abmeldungen, Transaktionen, Datenübertragungen,…) lückenlos aufgezeichnet?
- Haben Unbefugte keinen Zugriff auf die Protokolldateien?
- Können Administratoren die Protokollierung nicht unbemerkt manipulieren oder deaktivieren?

Zugriffsrechte
- Können Zugriffsrechte (auf Server, Datenbanken, Firewall, Dateien,…) nur von berechtigten Personen vergeben und geändert werden?
- Werden alte Zugriffsrechte gesichert und können sie zurückgespielt werden?
- Ist sichergestellt, dass Änderungen von Daten (zum Beispiel in der Datenbank) ausschließlich von autorisierten Personen vorgenommen werden können?
- Können Zugriffsrechte nicht über das Web administriert werden?

Penetrationstest
- Wurde ein Penetrationstest von Spezialisten durchgeführt?
- Wurden alle potentiellen Schwachstellen getestet und ggf. beseitigt?
- Wurden folgende Angriffsversuche auf den/die Server erkannt und abgewehrt?
  - Brute Force Angriff
  - Denial of Service Angriff (DoS)
  - Man-in-the-Middle-Angriff (MitM)
  - SQL-Injektion

Cross Site Scripting (XSS)[GL]
- Sind die erlaubten Eingaben genau definiert und sind nur diese Eingaben möglich?
- Werden alle Benutzereingaben (Suchwerte, Formulareinträge, Forenbeiträge,…) auf unerlaubten Inhalt geprüft?
- Werden alle Metazeichen <, >, &," oder' durch Zeichenreferenzen ersetzt (zum Beispiel „&" durch „&")? *Achtung, die Metazeichen sind umgebungsabhängig (HTML, JavaScript, CSS, XML,…)*.

Ein schönes Beispiel für das Ausnutzen einer XSS-Lücke zeigt heise.de [URL: heise-XSS]. Hier wird geschildert, wie 2007 die Website der Bundesregierung mit fremden Bildern „verunstaltet" wurde.

9.10.3 Werkzeuge für den Sicherheitstest

Zur Unterstützung des Penetrationstests werden **Security Test Tools** (Sicherheitskonfigurations- und Protokollierungswerkzeuge) eingesetzt. Sie überprüfen die Systemkonfiguration und suchen nach Schwachstellen, wie zum Beispiel fehlenden Passwortabfragen

oder offenen Ports, die einen unerlaubten Zugang zum Webserver ermöglichen. Eine Liste von kommerziellen Werkzeugen zur Web-Sicherheit ist unter [URL: TestToolsSQA] in der Rubrik „Web Site Security Test Tools“ zu finden. Ebenso bei [URL: TestToolsOPEN-SOURCE] in Rubrik „Testing Tools – Security“.

Das Bundesamt für Sicherheit in der Informationstechnik (BSI) unterstützt das Open Source Framework für Schwachstellen-Scanning und Schwachstellen-Management **OpenVAS**(Open Vulnerability Assessment System), s. [URL: OpenVAS].

9.10.4 Empfehlungen zum Sicherheitstest

Sicherheitstests auch in der Betriebsphase
Der Sicherheitstest sollte schon in frühen Projektphasen mit der Analyse der Sicherheitsvorkehrungen beginnen, während der Penetrationstest in der Teststufe Abnahmetest und/oder im Pilotbetrieb in der produktiven Systemumgebung stattfindet. Die Überprüfung der Sicherheit darf während der gesamten produktiven Phase des Anwendungssystems nicht aus den Augen verloren werden, weil aufgrund der sich permanent verändernden Internettechnologien neue Sicherheitslücken entstehen können.

„Legale Hacker“ engagieren
Kritische Apps müssen einem intensiven Sicherheitstest unterliegen, der entsprechend aufwendig und teuer ist. Weil für die Durchführung von Sicherheitstests Spezialistenwissen notwendig ist, kann es effektiver sein, „legale Hacker“ zu engagieren, als die Tests mit eigenen Kräften durchzuführen.

Bei weniger kritischen Anwendungen ist der Einsatz der oben abgebildeten Checkliste „Sicherheitstest“ sehr hilfreich und kostengünstig.

Weiterführende Informationen zum Thema
Das Bundesamt für Sicherheit in der Informationstechnik (**BSI**) [URL: BSI] hat einen Maßnahmenkatalog und Best Practices Sicherheit von Webanwendungen erstellen lassen und veröffentlicht [BSI-SecureNet_2006].

Eine wichtige Informationsquelle zum Thema Sicherheit ist das **Open Web Application Security** Project (**OWASP**) [URL: OWASP].

✓ Checkliste Firewall (extern)
Der Landesbeauftragte für den Datenschutz Niedersachsen (**LFDN**) stellt zum Thema Datenschutz/Datensicherheit eine Firewall-Checkliste zur Verfügung [URL: LFDN-Firewall]. Darin werden datenschutzrechtliche Anforderungen, technische und organisatorische Maßnahmen gegen Angriffe auf das Firewall-System und aus dem Internet auf das gesicherte Netz sowie Sicherungsmaßnahmen zur Paketfilterung behandelt.

…und vor dem Sicherheitstest alle Daten sichern!

9.10.5 Qualitätsanforderungen zum Sicherheitstest

Der Erfüllungsgrad von 100 % der Checklisten „Sicherheitstest“ und „Firewall“ ist die erste Qualitätsanforderung an den Sicherheitstest.

Die zweite Anforderung ist, dass keine Person, insbesondere kein beauftragter Hacker, in das System eindringen konnte. Für diese Anforderung müssen in der Testspezifikation konkrete Testendekriterien festgelegt werden, wie zum Beispiel Dauer und Intensität der Hackerangriffe.

9.11 Interoperabilitätstest

Eine Intranet-Anwendung, die auf einer definierten Datenbank läuft und für die Betriebssysteme und Browser vorgeschrieben sind, muss nicht interoperabel mit weiteren Systemkomponenten sein. Glück für die Tester.

Eine Mobile-Web-App muss unter vielen Kombination von Betriebssystemen, Browsern, Datenbanken, Endgeräten, Dateneingangskanälen, Datenausgangskanälen, Sprachen und Nutzertypen funktionieren. Sie muss interoperabel sein und alle – oder zumindest viele – der möglichen Konfigurationsmöglichkeiten müssen getestet werden. Pech für die Tester.

Interoperabilität ist die Fähigkeit eines Softwareprodukts, mit einer oder mehreren spezifizierten Komponenten zusammenzuwirken. Die Interoperabilität stellt die gemeinsame Nutzung von Daten, Programmen und Prozessen sicher.

► Der **Interoperabilitätstest** stellt die Interoperabilität eines Softwareprodukts für die für sie vorgesehenen Systemkonfigurationen sicher.

Zum Interoperabilitätstest gehören der Plugin-Test, der im Kap. 9.7 detailliert beschrieben ist, sowie die im Folgenden ausgeführten Cross-Browser-Tests und Cross-Device-Tests. Diese sind besonders wichtig, wenn mobile Endgeräte im Fokus der Anwendung stehen.

Um beim Interoperabilitätstest die Anzahl der zu testenden Systemkonfigurationen im Zaum zu halten, werden die Methoden des Paarweisen Testens (s. Kap. 5) eingesetzt.

9.11.1 Cross-Browser-Test

Browser von unterschiedlichen Herstellern reagieren nicht identisch. Das Erscheinungsbild und das Verhalten einer Website oder Web-App können sich von Browser-Typ zu Browser-Typ und im Zusammenspiel mit unterschiedlichen Betriebssystemen ändern. Daher muss eine browserbasierte Anwendung je nach Zielgruppe mit verschiedenen Browsern in unterschiedlichen Versionen getestet werden.

► Der **Cross-Browser-Test** überprüft die korrekte Darstellung, Funktionalität und Bedienbarkeit eines browserbasierten Testobjekts mit unterschiedlichen Browser-Konstellationen.

Es stellt sich die Frage, mit welchem Browser in welcher Version das Testobjekt getestet werden muss. Die Antwort hängt von der gewünschten Reichweite der zu testenden Anwendung ab. Ist eine Website oder eine Web-App nur einem eingeschränkten Nutzerkreis im Intranet zugänglich, so ist der eingesetzte Browser bekannt oder sogar verbindlich vorgeschrieben. In diesem Fall muss die Anwendung nur für diesen Browser getestet werden.

Ist eine Website im Internet frei zugänglich, so muss der Test auf alle gängigen Browser und Betriebssysteme erweitert werden.

9.11.2 Cross-Device-Test

Besonders spannend sind Interoperabilitätstests, wenn noch alle möglichen Smartphones und Tablet-PCs in die zu testenden Konfigurationen mit einbezogen werden müssen.

► Im **Cross-Device-Test** wird die Interoperabilität einer Software mit den spezifizierten Endgeräten getestet, d. h. die unterschiedliche Hardware, unter denen die Software eingesetzt werden soll, wird in die zu testenden Systemkonfigurationen aufgenommen.

Der Cross-Browser-Test hat als Schwerpunkt das Ziel, die Darstellung und das korrekte Verhalten einer Anwendung in möglichst vielen – auch mobilen – Browsern sicherzustellen.

Beim Cross-Device-Test liegt der Schwerpunkt der Tests auf der Kommunikation einer Mobile-App mit dem Backend-System. Besonders für Apps, die anwendungsbedingt regelmäßig Daten mit Servern austauschen und nicht nur für genau einen mobilen Endgerätetyp bzw. einen Endgerätehersteller entwickelt wurden, ist der Interoperabilitätstest mit unterschiedlicher Hardware wichtig.

Die Grenzen zwischen den beiden Testarten Cross-Browser- und Cross-Device-Test sind für Mobile-Web-Apps fließend.

9.11.3 Vorgehen beim Interoperabilitätstest

Folgendes Vorgehen ist beim Interoperabilitätstest praktikabel:

1. **Statistiken besorgen**
Zu aktuell verwendeten Systemkomponenten (Betriebssysteme, Browser, Endgeräte, Datenbanken,…) werden Statistiken besorgt[11] oder rechtzeitig im Internet beauftragt oder selbst erstellt[12].

[11] Zum Beispiel Browserstatistiken bei [URL: Browser-Statistik].

[12] Zum Beispiel mit einem User Tracking Tool, s. Abschn. 10.3.6.

2. **Risikoklassen für Komponenten festlegen**
Risikoklassen für die einzelnen Systemkomponenten werden auf Basis der Statistiken festgelegt.

3. **Kombinationen bestimmen und mit Risikoklassen versehen**
Die zu testenden Kombinationen von Systemkomponenten werden mit den Methoden des Paarweisen Testens (s. Kap. 5) und einem geeigneten Tool festlegt und erhalten anschließend auf Basis der Risikoklassen der einzelnen Systemkomponenten ebenfalls Risikoklassen.

4. **Testtiefe und Testfälle für die zu testenden Kombinationen festlegen**
Die Testtiefe und die Testfälle werden für die zu testenden Kombinationen auf Basis der Risikoklassen festlegt. Je höher das Risiko einer Kombination ist, desto zahlreicher und intensiver müssen die Tests durchgeführt werden. D.h., es muss entschieden werden, welche Testarten (Cookie-Test, Plugin-Test, Sicherheitstest, Test der Browser-Einstellungen, Oberflächentest,…) mit welcher Intensität für eine Systemkonfiguration durchgeführt werden müssen.

5. **Interoperabilitätstests durchführen**
Bevor der Interoperabilitätstest in aller Breite durchgeführt wird, muss der Test in die Tiefe abgeschlossen sein. Das bedeutet, dass Funktionstests und Usability-Tests für die wichtigste bzw. die wichtigsten Systemkonfigurationen mit realen Endgeräten erfolgreich beendet wurden. Erst wenn diese Tests keine Fehlerwirkungen mehr hervorbringen, werden alle anderen Systemkonfigurationen getestet, denn wir dürfen nicht die Intention des Interoperabilitätstests vergessen:

► Wir suchen beim Interoperabilitätstest keine Fehler, sondern wir wollen uns versichern, dass sich alle Konfigurationselemente miteinander vertragen!

Beispiel Interoperabilitätstest – Cross-Browser-Test

Im Folgenden wird ein mögliches Vorgehen bei der Bestimmung der Testszenarien für den Cross-Browser-Test unseres Finanzierungsrechners beschrieben. Die Anforderung für das Beispiel ist, dass der Rechner als Web-App für den Einsatz auf Desktop-Rechnern vorgesehen ist.

Die Kernfrage ist dann: Mit welchen Betriebssystem- und Browser-Versionen muss der Finanzierungsrechner getestet werden? Bei der Beantwortung helfen uns Verwendungsstatistiken, die Risikoanalyse (s. Kap. 6) und das Paarweise Testen (s. Kap. 5).

Die vier Tab. 9.1, 9.2. 9.3 und 9.4 sind aus diversen Browser-Statistiken zusammengestellt worden. Sie zeigen, welche Betriebssysteme, Browser-Typen, Browser-Versionen und Browser/Betriebssystem-Kombinationen Ende 2012, Anfang 2013 am häufigsten eingesetzt wurden.

Tab. 9.1 Betriebssysteme

Betriebssystem	Anteil in %
Windows 7	56
Windows XP	22
Windows Vista	7
Mac OS X	8
Windows 8	3

Tab. 9.2 Browser-Typen

Browser	Anteil in %
Firefox	32
Chrome	36
Safari	12
Internet Explorer	15
Opera	2

Tab. 9.3 Browser-Versionen

Browser	Version	Anteil in %
Firefox	19	28
Chrome	25	14
Safari	6	10
Internet Explorer	9	8
Safari	4	6
Internet Explorer	8	4
Safari	5	3
Opera	12	2
Internet Explorer	7	2
Internet Explorer	10	2

Tab. 9.4 Browser/Betriebssystem

Browser/BS	Win(%)	Mac OS X(%)
Firefox	45	8
Chrome	12	
IE	10	
Safari		8

Wir machen das Produktrisiko am aktuellen und geschätzten Markanteil der einzelnen Komponenten fest. Je häufiger eine Komponente verwendet wird, desto höher ist die ihr zugeordnete Risikoklasse.

Mit Blick auf die Tab. 9.1 fällt daher die Entscheidung für die zu testenden Betriebssysteme auf Mac OS X, Win 7 und Win XP. Weil Windows 8 neu[13] auf dem Markt ist

[13] Achtung, wir befinden uns im Frühjahr 2013!

Tab. 9.5 Risikoklassen Browser

Browser-Typ	Version	Begründung	Risikoklasse
Firefox	19	Laut Statistik über 10 % Anteil	A
	20	Aktuelle Version	A
	21	Liegt als Betaversion vor	B
Chrome	25	Laut Statistik über 10 % Anteil	A
	26	Aktuelle Version	A
Safari	6	Laut Statistik 10 % Anteil	A
	4	Laut Statistik 5 %-10 % Anteil	B
Internet Explorer	9	Laut Statistik 5 %-10 % Anteil	B
	10	Aktuelle Version	B
Opera	12	Anteil unter 5 %, ein Opera-Vertreter soll sicherheitshalber getestet werden	C

und sicherlich mehr Marktanteile gewinnen wird, soll es ebenfalls getestet werden. Somit bekommen die drei ersten Betriebssysteme die Klasse A und Windows 8 die Klasse B. Linux ist zwar nicht in den Betriebssystemstatistiken aufgeführt, aber mit mindestens einem Linux-Derivat wollen wir unsere Website testen, zum Beispiel Ubuntu. Hier wird die Risikoklasse C vergeben.

Browser-Versionen kommen auf jeden Fall in den Test (s. Tab. 9.5), wenn sie entweder in der Statistik über 5 % Nutzungsanteil haben oder aktuell bzw. absehbar von neuen Versionen abgelöst werden und eine größere Anzahl von Installationen erwarten lassen. Browser-Versionen mit 10 % und mehr Nutzungsanteil werden der Risikoklasse A zugeordnet. Bei einem Nutzungsanteil zwischen 5 und 10 % wird die Risikoklasse B vergeben. Sollen weitere Browser-Versionen in den Test aufgenommen werden, erhalten diese die Risikoklasse C.

Kommen wir zu den zu testenden Browser/Betriebssystem-Paarungen. Die in Tab. 9.4 aufgeführten Kombinationen sind die zum Testzeitpunkt wichtigsten und sollen natürlich getestet werden. Diese Tabelle sagt aber nichts zu den Versionen der Komponenten aus und es fehlen der Opera-Browser und Linux-Betriebssysteme.

Wollen wir nun alle möglichen Kombinationen von Browsern und Betriebssystemen testen? Sicher nicht. Auch wenn wir berücksichtigen, dass Safari-Browser nur auf Mac-Rechnern und der Internet Explorer nur auf Windows-Systemen getestet werden sollen, hätten wir 40 Paarungen zu testen, viel zu viele. Und wenn wir schon dabei sind, möchten wir den Finanzierungsrechner in den drei Sprachen Englisch, Deutsch und Spanisch testen. Macht 120 Kombinationsmöglichkeiten. Besser bedienen wir uns der Methode des Paarweisen Testens aus Kap. 5 und eines kleinen Kniffs zur weiteren Reduzierung der Testkonstellationen.

Tab. 9.6 Paare ohne Versionen

Testfall	Browser	Betriebssystem	Sprache
1	Firefox	Windows 7	Englisch
2	Firefox	Windows XP	Deutsch
3	Firefox	Windows 8	Spanisch
4	Firefox	Mac OS X	Englisch
5	Firefox	Linux	Englisch
6	Chrome	Windows 7	Deutsch
7	Chrome	Windows XP	Englisch
8	Chrome	Windows 8	Englisch
9	Chrome	Mac OS X	Spanisch
10	Chrome	Linux	Deutsch
11	Safari	Mac OS X	Deutsch
12	IE	Windows 7	Spanisch
13	IE	Windows XP	Englisch
14	IE	Windows 8	Deutsch
15	Opera	Windows 7	Englisch
16	Opera	Windows XP	Spanisch
17	Opera	Windows 8	Deutsch
18	Opera	Mac OS X	*Englisch*
19	Opera	Linux	Spanisch
20	Safari	*Mac OS X*	Englisch
21	Safari	*Mac OS X*	Spanisch

Zuerst werden die Kombinationen aus Browser-Typen, Betriebssystemen und Sprachen gebildet. Nach dem Pairwise-Verfahren, das hier mit dem Tool Hexawise[14] angewendet ist, ergeben sich 21 Kombinationen (Tab. 9.6).

Anschließend werden die Browser-Versionen auf die Kombinationen verteilt und zwar gewichtet nach ihrer Reihenfolge in der Tab. 9.5. Das Ergebnis ist in Tab. 9.7 dokumentiert. Die Risikoklassen der Testfälle werden in dem Beispiel anhand der Risikoklassen der Betriebssysteme und Browser festgelegt, wobei die geringere Klasse gewinnt. Zum Beispiel hat in Kombination 3 der Firefox-Browser die Klasse A und Windows 8 die Klasse B, womit die Kombination die Risikoklasse B erhält.

Bevor der eigentliche Test starten kann, müssen die Maßnahmen zu den einzelnen Risikoklassen (s. Tab. 9.8) definiert werden, denn je größer das Risiko ist, desto intensiver müssen die Tests sein – wie wir in der Risikoanalyse im Kap. 6 gelernt haben.

Dieses Beispiel zeigt, wie die Systemkomponenten mit der Risikoanalyse und dem Paarweisen Testen für den Cross-Browser-Test zusammengestellt werden können. Aber wir können die Sache noch steigern.

[14] Hexawise s. Kap. 5.3.

Tab. 9.7 Paare mit Versionen

Testfall	Browser	Betriebssystem	Sprache	Risikoklasse
1	Firefox 19	Windows 7	Englisch	A
2	Firefox 20	Windows XP	Deutsch	A
3	Firefox 19	Windows 8	Spanisch	B
4	Firefox 20	Mac OS X	Englisch	A
5	Firefox 21	Linux	Englisch	C
6	Chrome 25	Windows 7	Deutsch	A
7	Chrome 26	Windows XP	Englisch	A
8	Chrome 25	Windows 8	Englisch	B
9	Chrome 26	Mac OS X	Spanisch	A
10	Chrome 25	Linux	Deutsch	C
11	Safari 6	Mac OS X	Deutsch	A
12	IE 9	Windows 7	Spanisch	B
13	IE 10	Windows XP	Englisch	B
14	IE 9	Windows 8	Deutsch	B
15	Opera 12	Windows 7	Englisch	C
16	Opera 12	Windows XP	Spanisch	C
17	Opera 12	Windows 8	Deutsch	C
18	Opera 12	Mac OS X	*Englisch*	C
19	Opera 12	Linux	Spanisch	C
20	Safari 4	*Mac OS X*	Englisch	B
21	Safari 6	*Mac OS X*	Spanisch	A

Tab. 9.8 Maßnahmen zu Risikoklassen

Risikoklasse	Maßnahmen	Testumgebung
A	• Funktionstest mit den *drei wichtigsten* Use Cases • Plugin-Test • Sicherheitstest • Test der Browser-Einstellungen – Checkliste „Browser-Einstellungen“ • Kompletter Oberflächentest – Checkliste „Oberflächentest“	Reale Browser-Installation
B	• Funktionstest mit *einem* der drei wichtigsten Use Cases • Plugin-Test • Sicherheitstest • Test der Browser-Einstellungen – Checkliste „Browser-Einstellungen“ • Stichproben aus dem Oberflächentest – Checkliste „Stichprobe Browser-Test“	Reale Browser-Installation
C	• Funktionaler Stichprobentest • Stichproben zum Sicherheitstest • Sichttest der Oberfläche	Browser-Simulationen

Tab. 9.9 BS-Versionen inkl. Mobile-BS

Betriebssystemversion	Anteil in %
Windows 7	45
Windows XP	18
Windows Vista	8
iPhone 6	*5*
iPad 6	*5*
Android 4.1	*3*
OS X 10.8	3
OS X 10.6	2
Android 2.3	*2*
Linux	2
OS X 7	2

Beispiel Interoperabilitätstest – Cross-Device-Test

Soll unser Finanzierungsrechner auch als Mobile-Web-App für mobile Geräte realisiert werden, müssen die mobilen Betriebssysteme, die mobilen Browser und die unterschiedlichen Endgeräte, auf denen sie lauffähig sind, mit in den Cross-Browser-Test aufgenommen werden. Diese Anforderung katapultiert uns direkt in den Cross-Device-Test.

Um die mobilen Komponenten mit in den Test einzubeziehen, können zum Beispiel die beiden Statistiken[15] aus Tab. 9.9 und 9.10 zu Rate gezogen werden. Sie zeigen die Anfang 2013 im deutschsprachigen Raum am häufigsten eingesetzten Betriebssystem- und Browserversionen ab 2 % Marktanteil inklusive der Mobile-Versionen.

Als nächstes müssen die zu testenden physikalischen Endgeräte eruiert und mit in die Kombinatorik der Systemelemente aufgenommen werden. Spätestens hier freuen wir uns über den Einsatz der Risikoanalyse und eines Tools zum Paarweisen Testen…

Lassen wir es an dieser Stelle gut sein, damit das Beispiel zum Interoperabilitätstest nicht zu groß und langweilig wird. Schließen wir es an dieser Stelle mit den folgenden Erkenntnissen ab:

1. Je nachdem, wer die Statistiken erhoben hat und wo sie regional erhoben wurden, unterscheiden sich die Ergebnisse und somit die durchzuführenden Testszenarien. Daher muss genau geprüft werden, ob die Statistiken der gewünschten Zielgruppe entsprechen. Die Erhebung eigener Statistiken kann daher von Vorteil sein.
2. Der Testaufwand erhöht sich massiv, wenn eine Website bzw. App für den mobilen Einsatz gedacht ist. Die möglichen und zu testenden Kombinationen Client-Betriebssystem/Browserversion/Sprache/Endgerät-Hardware/Server-Datenbank/Server-Betriebssystem/SIM-Karten/Bandbreiten/… erhöhen sich exponentiell, daher sind hier die Methoden und Tools der Risikoanalyse und des Paarweisen Testens Gold wert.

[15] Diese beiden Statistiken basieren auf Veröffentlichungen auf [URL: Webmasterpro].

Tab. 9.10 Browserversionen inkl. Mobile

Browser	Version	Anteil in %
Firefox	20.0	32
Chrome	26.0	13
Internet Explorer	9.0	8
Mobile Safari	*6.0*	*8*
Internet Explorer	8.0	7
Android Browser	*4.8*	*6*
Internet Explorer	10.0	6
Safari	6.0	3
Opera	12.1	2
Safari	5.1	2
Safari	5.0	2

Abb. 9.13 browsershots

9.11.4 Werkzeuge für den Interoperabilitätstest

Tools zur Methodik

Tools zum Paarweisen Testen (s. Kap. 5) werden im Interoperabilitätstest eingesetzt, um effizient die zu testenden Systemkonfigurationen zu ermitteln.

Im Internet sind zahlreiche Anbieter von **User Tracking Tools** (s. Abschn. 10.3.6) zu finden. Diese Werkzeuge liefern neben Auswertungen zum Benutzerverhalten viele Statistiken, wie zum Beispiel zu Spracheinstellungen, eingesetzten Browser-Versionen, Betriebssystemen, Plugins und mobilen Geräten.

Tools zum Browsertest

Es gibt Online-Dienste, mit denen **statische Screenshots** von einzelnen Webseiten mit unterschiedlichen Browsern unter verschiedenen Betriebssystemen gemacht werden, wie zum Beispiel browsershots.org [URL: BrowserSnapshot], s. Abb. 9.13. Ein solcher stati-

Abb. 9.14 VirtualBox

scher Cross-Browser-Check eignet sich in der Anfangsphase eines Browsertests und für einen Sichttest, wie er im obigen Beispiel für die Testfälle der Risikoklasse C gefordert ist.

Auf spoon.net [URL: BrowserSandbox] kann der Cross-Browser-Test **dynamisch in einer Sandbox** durchgeführt werden. Dieser Dienst bietet via Plugin den Test mit unterschiedlichen Desktop- und Mobile-Browsern in virtuellen Umgebungen an, in einer Basisversion auch kostenlos.

Wer für seine Tests alte Browser-Versionen sucht, wird im **Browser-Archiv** [URL: Browser-Archiv] fündig.

Zum Nachweis der Browser-Kompatibilität einer Webapplikation ist zum Beispiel das Open Source Tool **Selenium** [URL: Selenium] geeignet, weil die Testskripte unter mehreren Betriebssystemen und Browsern eingesetzt werden können.[16]

Für Cross-Browser-Tests von mobilen Websites werden im Netz **Simulatoren** [GL] und **Emulatoren** [GL] angeboten, s. [URL: Emulatoren].

Technische Unterstützung der Tests

Weil nicht alle Konfigurationen für einen umfangreichen Interoperabilitätstest real zur Verfügung gestellt werden können oder müssen, werden viele in **virtuellen Maschinen** wie VirtualBox [URL: virtualbox] und VMware-Player [URL: vmware] zum Test bereitgestellt. In den virtuellen Maschinen können Web-Apps mit verschiedenen Browsern unter unterschiedlichen Betriebssystemen im Cross-Browser-Test getestet werden, wie z. B. mit Firefox 16 unter Linux Mint in einer VirtualBox (s. Abb. 9.14).

Auch für Cross-Device-Tests mit mobilen Endgeräten stellen die Hersteller dieser Geräte oftmals Emulatoren zur Verfügung, die schon für einige virtuelle Maschine vorkonfiguriert sind. In Abb. 9.15 ist zum Beispiel der Emulator von BlackBerry 10 im VMware-Player installiert.

Cross-Browser- und Cross-Device-Tests müssen bei Erscheinen neuer Versionen von Browsern, Betriebssystemen, Endgeräten wiederholt werden, um sicherzustellen, dass die

[16] Selenium siehe auch Abschn. 13.4.2.

Abb. 9.15 VMware-Player

Webanwendung benutzbar bleibt. Daher lohnt es sich, die Tests mit einem **Capture Replay Tool** (s. Kap. 14.3) zu automatisieren. Mit „guten" Tools können Tests mit einem Browser-Typ aufgezeichnet und mit einem anderen Browser-Typ wieder abgespielt werden.

9.11.5 Qualitätsanforderungen zum Interoperabilitätstest

Die Qualitätsanforderungen zum Interoperabilitätstest leiten sich aus den für die Testszenarien festgelegten Risikoklassen sowie den vorgegebenen Checklisten ab. Hier ein Beispiel für den Cross-Browser-Test:

- Alle mit Risiko A bewerteten Testszenarien sind bezogen auf die Checklisten „Browser-Einstellungen" und „Oberflächentest" zu 100 % fehlerfrei getestet. Es gibt keine offenen Fehler zum Test der für diese Risikoklasse festgelegten Use Cases, zum Sicherheits- und zum Plugin-Test.
- Alle mit Risiko B bewerteten Testszenarien sind bezogen auf die Checklisten „Browser-Einstellungen" und „Browser-Stichprobentest" zu 100 % fehlerfrei getestet. Es gibt

keine offenen Fehler der Priorität A oder B zum Test des für diese Risikoklasse festgelegten Use Case.
- Alle mit Risiko C bewerteten Testszenarien haben beim Stichprobentest und Sichttest der Oberfläche keine offenen Fehler der Priorität A und B.

9.11.6 Empfehlungen zum Interoperabilitätstest

Fremdsprachige Browser-Versionen testen
In den Anforderungen an eine Anwendung muss festgelegt werden, welche fremdsprachigen Browser- und Betriebssystemversionen für den Betrieb relevant sind und in den Test mit aufgenommen werden müssen.

Release Notes lesen
Eine Website oder App muss nicht unbedingt für jedes Release und für jede Version eines Browsers erneut getestet werden. Der Testdesigner sollte sich nach den Unterschieden erkundigen und die Release Notes lesen. Erst dann entscheidet er die notwendigen Testmaßnahmen. Das gleiche gilt für neue Versionen von Betriebssystemen und mobilen Endgeräten.

Technologien abwägen
Das Risiko, dass Darstellungen oder Funktionen einer Webanwendung empfindlich gegenüber Browsern sind, hängt auch von den eingesetzten Webtechniken ab. Ein geringes Risiko besteht zum Beispiel bei Verwendung von statischen HTML-Seiten und JavaScripting, ein hohes beim Einsatz von Java-Applets oder von ActiveX[GL]. Der Einsatz von bestimmten Technologien sollte daher mit Blick auf mögliche Risiken und den entstehenden Testaufwand genau überlegt werden.

Im Betrieb weiter testen
Plugin-, Cross-Browser- und Cross-Device-Tests müssen regelmäßig während des laufenden Betriebes wiederholt werden, weil sich die Systemlandschaft stetig verändert und somit Regressionstests notwendig werden (s. Kap. 14.2). Für jede Testwiederholung müssen die Systemkonfigurationen überarbeitet werden. Dazu ist es notwendig, dass stets aktuelle Statistiken vorliegen, die aufzeigen, mit welchen Browsern, Betriebssystemen und Endgeräten die Nutzer arbeiten. Das alles spricht für eine Automatisierung der Tests.

9.12 Mobile-Funktionstest

Mobile Endgeräte sind schon bei den Testarten der vorangehenden Kapitel – insbesondere beim Interoperabilitätstest – erwähnt und berücksichtigt. In diesem Kapitel werden Besonderheiten beim Funktionstest von „mobilen Testobjekten“ betrachtet; dazu gehören

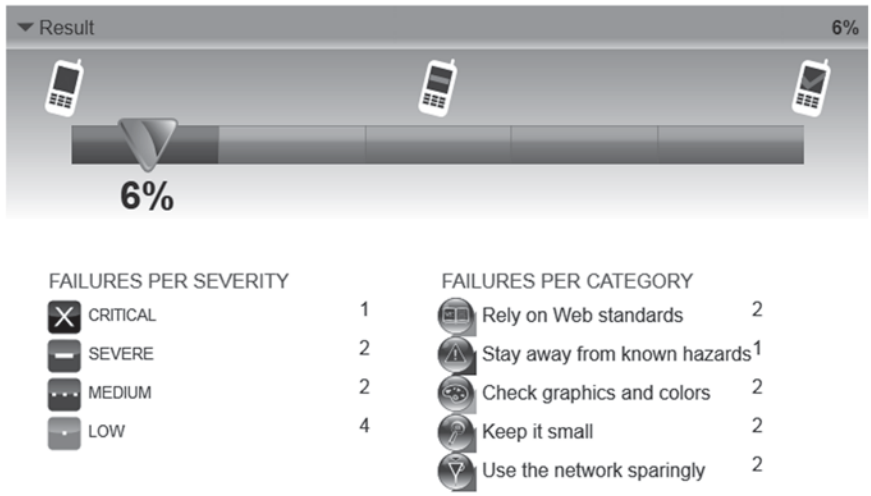


Abb. 9.16 mobileOK-Check Desktop-Webseite

das Qualitätsmerkmal Mobiltauglichkeit (s. Abschn. 2.2.2), die Erkennung mobiler Geräte und die besonderen Funktionalitäten, die mobile Geräte auszeichnen.

► Der **Mobile-Funktionstest** stellt die Funktionalität von Software-Komponenten für den Einsatz auf mobilen Endgeräten sicher.

Alle Testobjekttype aus dem mobilen Umfeld – Desktop-Website, Mobile-Webseite, (Mobile-)Web-App – haben gemeinsam, dass für sie funktionale Testfälle nach den Testentwurfsverfahren in Kap. 4 geschrieben werden. Diese fachlichen Testfälle müssen um „mobile Testfälle" ergänzt werden, die auf speziellen Fähigkeiten der Mobilgeräte beruhen.

9.12.1 Prüfung auf Mobiltauglichkeit

Soll ein Web-Auftritt, der Inhalte für Desktop-Benutzer publiziert, auch auf mobilen Endgeräten funktionieren, genügt es, wenn stichprobenhaft Testfälle aus dem Komponententest ausgewählt und mit einigen mobilen Endgerättypen geprüft werden.

Aber bevor der Funktionstest begonnen wird, ist es hilfreich zu prüfen, ob die zu testenden Webseiten generell **mobiltauglich** sind. Eine solche Prüfung kann zum Beispiel mit dem **W3C mobileOK Checke** [URL: W3C-validatorMobileOK] automatisiert werden. Abb. 9.16 zeigt das Prüfergebnis einer Desktop-Webseite mit mobileOK-Check. Die gefundenen kritischen und schweren Mängel sollten behoben werden.

Was für eine Desktop-Webseite optional ist, ist für eine Mobile-Webseite verpflichtend, sie *muss* mobiltauglich sein. Abb. 9.17 zeigt das Prüfergebnis einer Mobile-Webseite mit mobileOK-Check. Wenn das Prüfergebnis nicht den festgelegten Qualitätsanforderungen (s. Abschn. 9.12.6) entspricht, wird erst gar nicht mit dem Funktionstest begonnen.

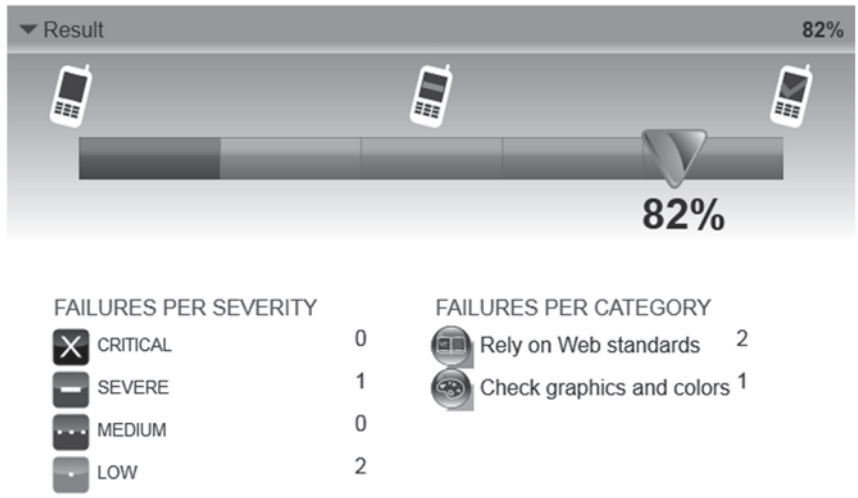


Abb. 9.17 mobileOK-Check Mobile-Webseite

Abb. 9.18 Mobile Sicht von w3c.de

9.12.2 Erkennung mobiler Geräte

Neben den üblichen fachlichen Funktionstests müssen eventuell zusätzliche Funktionen, die dem mobilen Einsatz dienen, getestet werden. Dazu gehören Geräteerkennung, Weiterleitung und Responsivität.

Mobile Sicht anzeigen
Ist ein kompletter Webauftritt oder zumindest ein bestimmter Teil für die mobile Präsentation vorgesehen, ist es schick, wenn die Website automatisch das Endgerät des Benutzers erkennt. Dann kann sie mit einem optimierten Stylesheet dargestellt werden.

Wird zum Beispiel die Website http://w3c.de mit einem mobilen Gerät aufgerufen, so wird automatisch auf die mobile Sicht der Website umgestellt, wie Abb. 9.18 zeigt.

Abb. 9.19 Responsive Design Viewer

Weiterleitung zur mobilen Webs(e)ite

Eine andere Möglichkeit ist, dass der Benutzer auf die mobile Version der Website oder ein spezielles Unterverzeichnis weitergeleitet wird. Beispiele für die Erkennung eines mobilen Endgeräts sind die Websites http://stern.de und http://zdf.de, die automatisch auf http://mobil.stern.de bzw. http://m.zdf.de umschalten.

Für den Nutzer komfortabel ist eine Website, die auch noch erkennt, um welchen Endgerättypen es sich handelt und daraufhin mit entsprechender Anzeige reagiert: BlackBerry, iPad, iPhone, Windows Phone…

Dass diese beschriebenen Weiterleitungen auch korrekt funktionieren, muss unter verschiedenen Konstellationen im Funktionstest anhand von „mobilen" Testfällen geprüft werden.

Responsive Websites

Noch eleganter verhalten sich **responsive** (reaktionsfähige) Websites die auf HTML5 und CSS3 basieren. Inhalte, Struktur und Navigationselemente reagieren auf das aufrufende Endgerät und passen sich automatisch an dessen Auflösung an. Ein Beispiel einer responsiven Website zeigt http://aids.gov. Die Darstellung dieser flexiblen Webseiten kann mit diversen Tools überprüft werden, eines davon ist der kostenlose „Responsive Design Viewer" [URL: Responsive Viewer], s. Abb. 9.19.

In der Medizin ist Responsivität (Ansprechempfindlichkeit) die Empfindlichkeit, mit der ein Organ auf einen äußeren Einfluss anspricht, zum Beispiel auf Medikamente[17]*. Sehr passend.*

[17] Oder auch Rotwein?!

Damit die Funktionalität bzgl. Der Erkennung mobiler Geräte gewährleistet ist, sind folgende Punkte mit Testfällen zu belegen:

✓ **Checkliste Erkennung mobiler Geräte**
- Erkennt die Website/Webseite korrekt, ob der Zugriff von einem mobilen Gerät aus erfolgt? Das bedeutet je nach Realisierung:
  - Wird automatisch auf die mobile Sicht der Website gewechselt?
  - Bzw. wird die Anfrage automatisch von der Desktop-Website auf die Mobile-Website umgeleitet?
  - Bzw. funktioniert die responsive Website korrekt?
- Kann der Benutzer manuell von der Mobile-Website auf die Desktop-Website umschalten?
- Kann der Benutzer manuell von der Desktop-Website auf die Mobile-Website umschalten?
- Sind alle wichtigen Funktionen der Desktop-Version auch in der mobilen Version der Website vorhanden?
- Ist die Navigation auf dem mobilen Gerät einwandfrei möglich?
- Werden möglichst keine Flash[GL]-Programme eingesetzt? (Diese funktionieren nicht auf allen gängigen Mobilgeräten.)

Funktionen wie Geräteerkennung und Weiterleitung müssen insbesondere im funktionalen Systemtest mit unterschiedlichen Browser- und Hardware-Konstellationen überprüft werden, d. h. umfangreiche Cross-Browser- und Cross-Device-Tests sind dazu notwendig (s. Kap. 9.11).

9.12.3 Spezielle Mobile-Funktionen

Mobile Geräte bieten besondere Funktionalitäten, die in Mobile-Webseiten und Mobile-Apps eingebunden werden können. Diese Integration muss natürlich getestet werden. Beispiele dafür sind, das Mobile-Tagging, die direkte Kontaktaufnahme mit dem Benutzer und die Integration von Kameraaufnahmen.

Mobile-Tagging
Beim Mobile-Tagging wird mit der Kamera des mobilen Gerätes ein Barcode von einem Objekt (Display, Schild, Zeitung…) ausgelesen. Eine solche spezielle Mobile-App entschlüsselt den Inhalt des Barcodes, der wiederum die weiteren Aktionen steuert.

Wird z. B. eine URL entschlüsselt, wird der Benutzer auf diese bestimmte Webseite weitergeleitet. Ein entschlüsselter Transaktionscode ruft einen Online-Dialog auf, ein Zugangscode erlaubt den Zugriff auf zusätzliche Daten oder eine Visitenkarte speichert die dekodierten Kontaktdaten auf dem Mobilgerät.

Abb. 9.20 QR-Code mit E-Mail

Als Beispiel ist im QR-Code[GL] in Abb. 9.20 eine E-Mail-Adresse verschlüsselt.[18]

Ziemlich „unglücklich" ist folgendes Beispiel

Ich scanne einen QR-Code. Die Entschlüsselung funktioniert einwandfrei und ich werde automatisch auf eine Webseite geleitet. Aber ich staune, denn ich lande auf einer ganz normalen Desktop-Webseite, die nicht für mobile Geräte gestaltet ist. Sie ist viel zu breit und hat viele Tabellenspalten. Schrift und Bilder sind viel zu winzig. Ich erkenne nichts auf dem Display meines Smartphones und bin sofort wieder weg von der Webseite.

So passiert mit einem QR-Code auf einem Werbebrief meiner Apotheke und – viel schlimmer – auch auf einer ganzseitigen Zeitschriftenanzeige eines großen Automobilkonzerns.

Hallo! Das Werbeinstrument Mobile-Tagging ist für Smartphones erfunden und muss auf einer mobiltauglichen Webseite landen!

Click to Call

Eine weitere mobile Spezialität sind Funktionen, die direkten Kontakt des Benutzers mit einer dritten Partei herstellen, wie zum Beispiel:

- Click to Call (CTC): Mit einem Klick wird direkt aus der Webseite heraus ein Telefongespräch zwischen einer dritten Partei und dem Benutzer eingeleitet, sofern er mit einem mobilen Telefon arbeitet.
- Real Callback: Der Besucher einer Website wird sofort, automatisch und (hoffentlich) kostenlos zurückgerufen.
- Click to SMS: Ein SMS-Dienst wird gestartet und automatisch wird eine SMS versendet. Click to SMS kann zum Beispiel für Abstimmungen genutzt werden.
- Click to Video Call: Ein Videotelefonat wird eingeleitet.

Kameraintegration

Soll eine Mobile-Web-App zum Beispiel aktuelle Kameraaufnahmen in die Anwendung integrieren, besteht die besondere Herausforderung des Funktionstests darin, dass diese

[18] Probieren Sie den QR-Code aus. Der Autor freut sich über eine E-Mail von Ihnen.

Funktionen unter allen für sie vorgesehenen Hardware-Umgebungen korrekt funktionieren.

Die drei Beispiele Mobile-Tagging, Click to Call und Kameraintegration zeigen, dass der Cross-Device-Test (s. Abschn. 9.11.2) besonders wichtig für Mobile-Apps ist.

9.12.4 Empfehlungen zum Mobile-Funktionstest

Störfälle
Beim funktionalen Systemtest einer Mobile-App müssen Unterbrechungen, wie z. B. eingehende Anrufe, E-Mails, SMS oder Terminbenachrichtigungen, definiert und geprüft werden. Diese und weitere Störtestfälle, die aufgrund von Netzwerkgegebenheiten oder Benutzeraktionen auftreten können, müssen unbedingt durchgeführt werden. Störtestfälle werden im Kap. 13.3 beschrieben, weil mit ihnen das Qualitätsmerkmal Zuverlässigkeit überprüft wird.

Mobiltauglichkeit vor Funktionalität vor Mobilfähigkeit
Soll eine Website oder App auf mobilen Endgeräten zum Einsatz kommen, sollte vor dem fachlichen Funktionstest die Prüfung auf Mobiltauglichkeit durchgeführt werden. Anschließend werden mit der wichtigsten Systemkonfiguration die funktionalen Testfälle ausgeführt. Danach beweisen Cross-Device-Tests (s. Abschn. 9.11.2), Mobile- Performanz-/Lasttests (s. Kap. 12.7) und Mobile-Zuverlässigkeitstests (s. Kap. 13.3) die Mobilfähigkeit des Gesamtsystems.

Wo kommen die vielen mobilen Endgeräte her?
Eine der großen Herausforderungen des Mobile-Funktionstests und insbesondere für den Interoperabilitätstest (s. Kap. 9.11) ist die große Anzahl der aktuellen physischen Endgeräte für den Test zu besorgen.

Wollen wir für den Test alle Geräte kaufen, Privatgeräte der Mitarbeiter nutzen oder gezielt externe Tester mit unterschiedlichen Geräten verpflichten? Wollen wir Cloud-Testing und/oder Crowd-Testing?

Diese strategische Entscheidung muss das Testmanagement auch für die Mobile-Tests zur Effizienz und Zuverlässigkeit treffen; dieses Thema wird deshalb im Abschn. 15.1.4 behandelt.

9.12.5 Werkzeuge für den Mobile-Funktionstest

Bevor Funktionstests mit Simulatoren und/oder „echter Hardware“ durchgeführt werden, muss die generelle Mobiltauglichkeit der Webseiten geprüft werden, zum Beispiel mit dem **W3C mobileOK Checker** [URL: W3C-validaterMobileOK]. Der W3C mobileOK

Checker liefert neben dem Prüfergebnis wie in Abb. 9.16 und 9.17 eine genaue Analyse, welche Fehlerursachen wo gefunden wurden und warum sie ein Problem darstellen. Auch wenn sich viele der Prüfpunkte auf die Benutzbarkeit der Webseite beziehen, ist das Ergebnis ein Kriterium dafür, ob der Funktionstest durchgeführt werden kann. Natürlich muss festgelegt sein, welchen Grad der Mobiltauglichkeit die Webseiten haben sollen, (s. Abschn. 9.12.6).

Für den Funktionstest einer Desktop- oder Mobile-Website müssen nicht immer reale Endgeräte eingesetzt werden; hier können **Simulatoren** und/oder **Emulatoren**[GL] ausreichen.

Für eine Mobile-App dagegen muss ein Systemtest auf den Geräten stattfinden, für die sie entwickelt wurde. Vorab können die Komponententests mit Simulatoren und/oder Emulatoren durchgeführt werden.

Für den Test einer Mobile-Web-App können nicht immer alle denkbaren Systemkonfigurationen real bereitgestellt werden; auch hier kommen Simulatoren und Emulatoren zum Einsatz.

Eine umfangreiche Aufstellung von Emulatoren und Simulatoren für den Mobile-Funktionstest gibt es auf [URL: Emulatoren].

9.12.6 Qualitätsanforderungen zum Mobile-Funktionstest

Testfälle
Die funktionalen Testfälle müssen wie immer zu einem definierten Prozentsatz erfolgreich abgeschlossen sein.

Für Mobile-Apps muss der funktionale Systemtest auf allen festgelegten Hardware/Betriebssystem-Konfigurationen fehlerfrei durchgeführt worden sein. Für Mobile-Web-Apps kommen zusätzlich noch die Browser-Versionen zu den zu testenden Kombinationen von Systemkomponenten hinzu (s. Abschn. 9.11.2 – Cross-Device-Test).

Mobiltauglichkeit
In Abb. 9.16 und Abb. 9.17 wurden Checks für Webseiten durchgeführt. Das Prüfergebnis zeigt, zu welchem Grad die Mobiltauglichkeit erreicht und wie viele kritische, schwere, mittlere und leichte Probleme erkannt wurden. Diese Maße eignen sich für „mobile“ Qualitätsanforderungen an eine Webseite, wie zum Beispiel:

- Eine Desktop-Webseite, die auch mobil angezeigt werden soll, darf keine kritischen Mängel haben.
- Eine Mobile-Website muss zu 80 % mobiltauglich sein und darf weder kritische noch schwere Mängel aufweisen.

9.13 Zusammenfassung

- Zu den Testarten, die als Schwerpunkt das Qualitätsmerkmal Funktionalität prüfen, gehören Unit-Test, Komponententest, Integrationstest, Systemtest, Link-Test, Cookie-Test, Plugin-Test, Webservice-Test, Sicherheitstest, Interoperabilitätstest und Mobile-Funktionstest.
- Der Unit-Test wird vom Entwickler parallel zur Programmierung durchgeführt.
- Wird eine Komponente im Rahmen des Unit-Tests nicht vollständig vom Entwicklerteam getestet, wird sie an das Testteam in den Komponententest für die restlichen Tests übergeben.
- Geprüfte Komponenten werden im Integrationstest zu Subsystemen zusammengeführt und getestet. Die Integration aller Subsysteme und Komponenten, die in mehreren Stufen erfolgen kann, bildet das Gesamtsystem.
- Der funktionale Systemtest testet die funktionalen Anforderungen des Gesamtsystems.
- Der Link-Test prüft neben der Funktionalität die Ordnungsmäßigkeit der gesetzten Links.
- Der Plugin-Test stellt die Funktionalität der benötigter Plugins und die Verträglichkeit der Webanwendung mit bereits installierten Plugins sicher.
- Webservices werden als selbständige Komponenten mit speziellen Testtreibern getestet.
- Die Sicherheit eines Anwendungssystems wird von Sicherheitsexperten im Penetrationstest überprüft.
- Bei „brisanten“ Apps sollten professionelle „Hacker“ versuchen, Sicherheitslücken aufzudecken.
- Der Interoperabilitätstest will keine Fehlerzustände finden, sondern nachweisen, dass unterschiedliche Systemkomponenten zusammen funktionieren.
- Zu den Interoperabilitätstests gehören Plugin-, Cross-Browser- und Cross-Device-Tests.
- Interoperabilitätstests werden zuerst in die Tiefe und dann in die Breite durchgeführt.
- Mobile-Apps und Webseiten müssen für den mobilen Einsatz mobiltauglich sein und auf mehreren Hardware-Konfigurationen getestet werden. Dabei können Simulatoren und Emulatoren zum Einsatz kommen.

Tests zur Benutzbarkeit

10

Nutzlose Objekte und nutzlose Subjekte lassen sich in drei Kategorien einteilen: Die einen arbeiten nicht, die anderen brechen zusammen und der Rest verschwindet einfach.

Russell Baker (*1925)

Die Benutzbarkeit ist für die Akzeptanz eines Webauftritts oder einer App elementar. Wenn der Benutzer eine Anwendung nicht erreichen, nicht mit seinem Standard-Browser ansehen, die Benutzerführung nicht verstehen oder seine gewünschten Informationen nicht abrufen kann, dann wechselt er zum Mitbewerber. Eine gute Anwendung zeichnet sich auch dadurch aus, dass sie intuitiv bedient und schnell erlernt werden kann.

Dieses Kapitel beschreibt die Testarten Content-, Oberflächen-, Usability-, Zugänglichkeits- und Auffindbarkeitstest, die das Qualitätsmerkmal Benutzbarkeit mit den Teilmerkmalen Bedienbarkeit, Erlernbarkeit und Verständlichkeit sowie den webspezifischen Merkmalen Auffindbarkeit, Barrierefreiheit und Rechtskonformität prüfen.

10.1 Content-Test

Es wird darauf hingewiesen, dass die folgenden Ausführungen zum Webrecht und zur Gesetzgebung nicht rechtsverbindlich sind und, ebenso wie die abgebildeten Checklisten, keinen Anspruch auf Vollständigkeit erheben.

Der Content-Test ist eine Ausprägung des Dokumententests (s. Kap. 8.2), der für Webauftritte von besonderer Bedeutung ist. Er wird in Form von Reviews oder schriftlichen Stellungnahmen anhand spezieller Checklisten durchgeführt.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_10

Eine Website ist ein Dokument, das sich einem weltweiten Benutzerkreis darbietet. Im Inhalt (Content) einer Website müssen unabhängig von fachlichen und technischen Funktionalitäten die besonderen Belange des WWW berücksichtigt werden. Dazu gehören die Erwartungen, die Benutzer an den Inhalt einer Website haben, die Aufklärungspflicht des Anbieters sowie die Rechtmäßigkeit des Informations- und Dienstleistungsangebotes.

► Der **Content-Test** stellt die Erfüllung der Benutzererwartungen und der Aufklärungspflicht sowie die Rechtskonformität einer Website sicher.

10.1.1 Erfüllung der Benutzererwartungen

Der Benutzer einer Website will sofort erkennen, ob die angebotenen Inhalte für ihn von Nutzen und aktuell sind. Er will mit der Anwendung intuitiv umgehen können, die gesuchten Informationen schnell erhalten und problemlos auf sie zugreifen können. Er hat konkrete Vorstellungen, welche Inhalte eine Website bieten muss. Dazu gehören:

- Detaillierte Informationen zu Produkten und Leistungen
- Kontaktdaten von Ansprechpartnern
- Aktuelle Unternehmensnachrichten
- Preise
- Tipps und Frequently Asked Questions (FAQ)
- Bestellmöglichkeiten

Auf Basis dieser Erwartungen und der von der ISO 9241 Teil 110 geforderten Qualitätsmerkmale Lernförderlichkeit und Individualisierbarkeit (s. Kap. 2.1) ergibt sich die Checkliste „Web-Content“:

✓ Checkliste Web-Content
Information
- Ist der Zweck der Website bzw. jedes Teilbereiches der Website deutlich erkennbar (kommerziell, informativ, unterhaltend)?
- Wird der Inhalt des Webangebotes seinem Zweck gerecht?
- Ist der persönliche Nutzen der Website für den Benutzer offensichtlich?
- Wird für jede Zielgruppe ein Mehrwert angeboten?
- Findet jede Zielgruppe schnell und ohne viel Aufwand ihre gewünschten Informationen?
- Erhält der Nutzer detaillierte Produkt- und Leistungsinformationen?
- Ist gewährleistet, dass die Inhalte der Website stets aktuell sind?
- Wird angezeigt, wann die Inhalte der Website zuletzt aktualisiert wurden?
- Ist eine Suchfunktion vorhanden?

- Gibt es folgende Rubriken:
  - FAQ?
  - Glossar?
  - Impressum?
  - Index?
  - News?
  - Tipps?
- Gibt es aktuelle Unternehmensnachrichten und Newsletter?
- Ist eine Bestellung von Informationsmaterial möglich?
- Sind Preise angegeben?
- Wird der Nutzer über den Verbleib seiner Eingabedaten informiert?

Kommunikation
- Sind die Kontaktdaten der Ansprechpartner angegeben?
- Sind für jede Nutzer-/Zielgruppe persönliche Ansprechpartner benannt?
- Ist die komplette Adresse des Anbieters angegeben?
- Gibt es einen Lageplan und eine Anfahrtsskizze?
- Ist eine E-Mail-Kontaktadresse angegeben?
- Werden Kommunikationsmöglichkeiten (Forum, Formulare, Gästebuch) bereitgestellt?
- Wird eine Hotline angeboten und ist deutlich erkennbar, unter welchen Bedingungen sie zu erreichen ist (Zeit, Preis, Medien)?
- Wird der Nutzer bei der Durchführung von Geschäftsprozessen über die nächsten Schritte informiert?

Internationalität
- Werden die wichtigsten Informationen in weiteren Sprachen angeboten?
- Werden Informationen in der jeweiligen Sprache der Zielgruppe angeboten?
- Entsprechen Maß- und Zeitangaben internationalen Standards?

Lernförderlichkeit
- Gibt es innerhalb der Hilfefunktion ein Stichwortverzeichnis?
- Gibt es innerhalb der Hilfefunktion eine Gliederung?
- Gibt es eine kontextsensitive Hilfe?
- Sind alle beschriebenen Hilfe- und Fehlertexte verständlich?
- Ist eine Guided Tour durch die Webapplikation vorgesehen?
- Ist die Guided Tour für die Zielgruppe verständlich?
- Können Testbuchungen/Testbestellungen vorgenommen werden?

Individualisierbarkeit – Persönliche Einstellungen
- Kann der Benutzer sich seine Inhalte individuell zusammenstellen, in seinem Profil speichern und wieder aufnehmen?
- Kann der Benutzer die Inhalte für seinen persönlichen Newsletter zusammenstellen?
- Kann der Newsletter abbestellt werden?

Content auf mobilen Geräten
- Sind Texte kurz und prägnant?
- Werden keine langen, zusammengesetzten Wörter benutzt?
- Ist die Lesbarkeit an den passenden Stellen durch Aufzählungspunkte optimiert?
- Ist die Anzahl der erforderlichen Schritte zur Durchführung der Geschäftsprozesse optimal klein gehalten? (Newsletter-Registrierung, Bestellung, …)
- Ist die Anzahl der Eingabefelder von Formularen optimal klein gehalten?
- Sind die Inhalte, die besonders für mobile Benutzer interessant sind, priorisiert und prominent dargestellt:
  - Adresse mit Anfahrtsbeschreibung, Karte und GPS-Daten?
  - Kontaktdaten?
  - Filialsuche?

Die Checkliste „Web-Content“ ist für Websites relevant, die einen hohen Informationsanteil besitzen und neue Benutzer binden möchten. Sie hat im Gegensatz zu den folgenden Themen der Aufklärungspflicht und der Gesetzeskonformität keinen verbindlichen Charakter.

10.1.2 Einhaltung der Gesetze

Eine Webanwendung muss rechtskonform sein, d. h. den Anforderungen des Gesetzgebers genügen. Die zu berücksichtigenden Gesetze sind in der Checkliste „Webrecht“ zusammengefasst.

✓ Checkliste Webrecht
Im Web zu beachtende deutsche Gesetze
- Bundesdatenschutzgesetz (BDSG)
- Dienstleistungs-Informationspflichten-Verordnung (DL-InfoV)
- Gesetz über die Verbreitung jugendgefährdender Schriften und Medieninhalte (GjS)
- Gesetz zum elektronischen Geschäftsverkehr (EGG)
- Gesetz zum Schutze der Jugend in der Öffentlichkeit (JÖSchG)

- Jugendmedienschutzstaatsvertrag (JMStV)
- Jugendschutzgesetz (JuSchG)
- Markengesetz (MarkenG)
- Signaturgesetz (SigG)
- Telemediengesetz (TMG)
- Staatsvertrag über Rundfunk und Telemedien (RStV)
- Telekommunikationsgesetz (TKG)[GL]
- Urheberrechtsgesetz (UrhG)
- Zugangskontrolldiensteschutz-Gesetz (ZKDSG)

Sonstige rechtliche Bedingungen
- Ist das Wettbewerbsrecht eingehalten?
- Ist das Presserecht eingehalten?
- Ist die Signaturverordnung (SigV) eingehalten?
- Sind Preise nach den Vorgaben der Preisangabenverordnung (PAngV) angegeben?
- Werden EU-Rechte eingehalten?
- Werden internationale Rechte eingehalten?

Informationen und Gesetzestexte zu den aufgeführten Gesetzen sind auf [URL: Gesetze-BMJ] nachzulesen.

Die ausführliche Darlegung der in der Checkliste aufgeführten Gesetze würde den Rahmen des Buches sprengen. Die Liste ist als Grundlage für die Rechtsexperten gedacht, die bei der Veröffentlichung eines Webangebotes die Einhaltung der Gesetze und Vorschriften überprüfen und nachweisen müssen.

10.1.3 Rechtskonforme Domain-Namen

Der rechtskonforme Webauftritt fängt mit der Namensgebung der Domain an, denn der darf keine Namen – auch nicht in Teilen oder Wortkombinationen – enthalten von:

- Gemeinden oder Städten
- Behörden, öffentlichen Diensten
- Fremden, markenrechtlich geschützten Produkten oder Begriffen
- Bekannten Persönlichkeiten
- Bekannten Fernsehsendungen
- Bekannten Publikationen (Bücher, Zeitungen, …)
- Personen, sofern keine Genehmigung der Namensnutzung vorliegt

Um sicherzugehen, ist eine Recherche unter anderem in Suchmaschinen, in Handelsregistern, bei Patent- und Markenämtern und Registrierungsstellen[1] von Domain-Namen unabdingbar.

10.1.4 Erfüllung der Aufklärungspflicht

Der Betreiber einer Website hat dem Benutzer gegenüber eine Aufklärungspflicht. Zudem muss ein Webauftritt rechtlich und für den Benutzer erkennbar abgesichert sein.

Am 1. März 2007 hat das Telemediengesetz das **Teledienstegesetz**, das Teledienstedatenschutzgesetz und den Mediendienste-Staatsvertrag abgelöst. Zu den darin beschriebenen Telemedien gehören auch Webseiten. Das Telemediengesetz verlangt nach § 5 **TMG** von allen gegen Entgelt angebotenen und geschäftsmäßig betriebenen Webauftritten Pflichtangaben in einem Impressum. Die Datenschutzbestimmungen sind im Wesentlichen vom zuvor gültigen Teledienstedatenschutzgesetz und dem Mediendienste-Staatsvertrag übernommen.

Bei journalistisch-redaktionell gestalteten Webinhalten, in denen insbesondere vollständig oder teilweise Inhalte periodischer Druckerzeugnisse in Text und Bild wiedergegeben werden, müssen neben den Angaben nach § 5 TMG auch der oder die Verantwortlichen mit Namen und Anschrift benannt werden (gemäß § 55 **RStV**).

Anbieter von Dienstleistungen müssen vor der Erbringung der Dienstleistung nach § 2 **DL-InfoV** bestimmte Informationen in klarer und verständlicher Form zur Verfügung stellen.

Wichtige Punkte zur Aufklärungspflicht von Website-Anbietern, die sich aus den oben aufgeführten Gesetzen ergeben, sind in der folgenden Checkliste „Aufklärungspflicht“ aufgeführt.

✓ Checkliste Aufklärungspflicht
Pflichtangaben für alle Webauftritte

- Sind die Pflichtangaben für natürliche Personen vollständig?
  - Familienname
  - Mindestens ein ausgeschriebener Vorname
  - Vollständige, ladungsfähige Postanschrift mit PLZ, Ort, Str., Hausnummer – *das Postfach ist nicht ausreichend!*
  - E-Mail-Adresse
  - Telefonnummer
  - Ggf. Faxnummer

[1] In Deutschland die DENIC eG [URL: denic]

- Sind die Pflichtangaben für juristische Personen bzw. Personengesellschaften vollständig?
  - Vollständiger und ausgeschriebener Firmenname
  - Rechtsform der Gesellschaft
  - Vertretungsberechtigte
  - Vollständige Firmenanschrift, ggf. die eigene Postleitzahl – *das Postfach ist nicht ausreichend!*
  - Hauptniederlassung, falls mehrere Niederlassungen existieren
  - Freiwillige Angaben über das Kapital der Gesellschaft (wenn, dann mit Stamm- bzw. Grundkapital und Gesamtbetrag der ausstehenden Einlagen)
  - E-Mail-Adresse
  - Telefonnummer
  - Ggf. Faxnummer

Zusätzliche Pflichtangaben für einen geschäftsmäßig betriebenen Webauftritt (gemäß § 5 TMG)

- Ist die Aufsichtsbehörde genannt, sofern der Dienst behördlich zugelassen werden muss (z. B. Gastronomiebetrieb, Makler)?
- Sind Register und Registernummer angegeben, sofern der Diensteanbieter in ein Register eingetragen ist (Genossenschafts-, Handels-. Partnerschafts- oder Vereinsregister)?
- Sind beim Handelsregister die Registernummer und das Registergericht angegeben?
- Wenn der Diensteanbieter einen reglementierten Beruf (z. B. Gesundheitshandwerk) oder einen Beruf, der besonderen Anerkennungsregeln unterliegt (z. B. Architekt), ausübt, sind dann angegeben:
  - Die berufsständische Kammer, der Berufsverband oder eine ähnlichen Einrichtung?
  - Die gesetzliche Berufsbezeichnung und der Staat, in dem die Berufsbezeichnung verliehen worden ist?
  - Die Bezeichnung der berufsrechtlichen Regelungen (z. B. Gesetzesbezeichnung)?
  - Der Zugang zu den berufsrechtlichen Regelungen (z. B. Website der betreffenden Kammer)?
- Ist die Umsatzsteueridentifikationsnummer oder die Wirtschaftsidentifikationsnummer angegeben, falls der Dienstanbieter eine solche besitzt?
- Falls sich die AG/KGaA/GmbH in einer anzugebenden Abwicklung oder Liquidation (TMG § 5 Nr. 7) befindet, sind die Angaben dazu veröffentlicht?

Zusätzliche Pflichtangaben für den Webauftritt mit journalistisch-redaktionellem Inhalt (gemäß § 55 RStV)

- Sind die Pflichtangaben gem. § 5 TMG vorhanden (s. o.)?
- Ist eine verantwortliche Person mit Namen und Anschrift benannt?
- Wenn mehrere Verantwortliche benannt sind, ist erkenntlich wer für welchen Dienst verantwortlich ist?

Pflichtangaben für den Webauftritt zur Erbringung von zahlungspflichtigen Dienstleistungen (gemäß § 2 DL-InfoV)

- Sind die wesentlichen Merkmale der Dienstleistung angegeben, soweit sich diese nicht bereits aus dem Zusammenhang ergeben?
- Sind die allgemeinen Geschäftsbedingungen (AGB) angegeben und offensichtlich zugänglich?
- Können die AGB gedruckt und gespeichert werden?
- Sind ggf. bestehende Garantien angegeben, die über die gesetzlichen Gewährleistungsrechte hinausgehen?
- Falls eine Berufshaftpflichtversicherung besteht, sind die Angaben zu dieser vorhanden, insbesondere Name und Anschrift des Versicherers und der räumliche Geltungsbereich?

10.1.5 Weitere rechtliche Angaben für Webauftritte

Neben den oben aufgeführten Pflichtangaben muss die Qualitätssicherung weitere Punkte aus rechtlicher Sicht prüfen.

Wenn vom Benutzer personenbezogene Daten verlangt werden (z. B. bei Newsletter-Anmeldung, Formulareingaben, …), ist ein **Datenschutzhinweis** auf der Website anzubringen, der erläutert, welche Daten für welchen Zweck gespeichert und wie sie konform zu Datenschutzgesetzen verwendet werden.

Ein Onlineshop muss **Angaben zur Kaufabwicklung** machen, d. h. korrekt und offensichtlich über Preise, Kosten und Fristen aufklären.

Copyrights müssen vorliegen und externe Links müssen rechtlich geprüft sein.

Um **Webanalysen** über das Benutzerverhalten zu einer Website zu erhalten, werden Tools eingesetzt, die IP-Adressen speichern. Das Speichern dieser IP-Adressen ist kritisch, weil darüber auf Nutzer geschlossen werden kann. Daher müssen die IP-Adressen anonymisiert werden. Der Benutzer muss darüber informiert werden, welche Analysetools eingesetzt werden, und er muss ein Widerspruchsrecht haben, das der Betreiber der Website umsetzt.

Gefällt-mir-Buttons zu sozialen Netzwerken wie Facebook, Google+ oder Twitter sind nicht datenschutzgerecht. Allein durch das Einbinden dieser Buttons und ohne dass der Benutzer auf diese geklickt hat, können via iFrames (s. Abschn. 9.5.1) und Cookies (s. Kap. 9.6) Daten zum Betreiber des sozialen Netzwerk gelangen und ausgetauscht werden. Wenn Gefällt-Mir-Buttons auf einer Website eingebunden werden, müssen dazu Datenschutzhinweise erfolgen und der Benutzer sollte mit sogenannten „**2-Klick-Buttons**“ (s. Abb. 10.1) die Möglichkeit haben, selbst zu bestimmen, ob und wann Daten an die sozialen Netzwerke übertragen werden.

Abb. 10.1 2-Klick-Buttons

Erst wenn der Benutzer einmal auf den „2-Klick-Button“ klickt, wird der Button aktiv und die Verbindung zum Server des Social-Networks hergestellt, und nur danach kann der Benutzer sein „Gefällt-mir“ übertragen. Diese Buttons sollten auch wieder deaktivierbar sein.

Informationen zur datenschutzkonformen Webanalyse gibt es zum Beispiel bei [URL: DS-Webanalyse01] und beim Bundesverband Digitale Wirtschaft (BVDW) e. V., der ein Whitepaper „Webanalyse und Datenschutz“ herausgebracht hat, [URL: DS-Webanalyse02].

In der folgenden Checkliste „Rechtliche Angaben“ werden die Prüfungspunkte zur Rechtskonformität zusammengefasst:

✓ Checkliste Rechtliche Angaben

Allgemeines

- Sind die Pflichtangaben unmittelbar erreichbar (z. B. durch einen gut sichtbaren Link „Impressum“ auf jeder Webseite)?
- Liegen alle Copyrights für Audio-Dateien, Bilder, Karten, Markenzeichen, Texte und Videos vor?
- Haben alle Seiten Copyright-Vermerke?
- Sind die Autoren der Website angegeben?
- Sind alle Links rechtlich geprüft (s. Checkliste „Link-Test“, Abschn. 9.5.1)?
- Sind alle Links mit Prüfer, Prüfdatum und Hardcopy dokumentiert?
- Existiert eine Abstandsklausel zu „verlinkten“ Inhalten?
- Werden keine fremden Markennamen in Metatags oder „unsichtbarer Schrift“[2], z. B. weiß auf weißem Grund, eingesetzt?
- Existiert eine Erklärung über die dargestellten Marken und Logos?
- Existiert der Haftungsausschluss bzgl. Aktualität, Korrektheit, Vollständigkeit und Qualität der bereitgestellten Informationen?
- Ist eine Absicherung gegenüber Downloads vorgenommen?

Sorgfaltspflicht

- Sind Werbung und sachliche Inhalte zweifelsfrei voneinander getrennt?
- Ist keine Schleichwerbung im Content enthalten?
- Sind Berichte und eigene Meinungen eindeutig ersichtlich und voneinander getrennt?
- Sind die fachlichen Inhalte von Fachleuten geprüft und freigegeben?

[2] By the way, unsichtbare Schrift fliegt bei einigen Suchmaschinen raus, was wiederum der Auffindbarkeit (s. Kap. 10.5) abträglich ist.

- Werden die Quellen zu den angebotenen Informationen genannt?
- Ist der Aufklärungspflicht Genüge getan (s. Checkliste „Aufklärungspflicht“, Abschn. 10.1.4)?

Angaben zu Verträgen
- Sind Nutzungs- und Vertragsbedingungen angegeben und offensichtlich zugänglich?
- Sind Preise bei jedem Bestellposten mit Mehrwertsteuer und mit allen zusätzlich anfallenden Kosten angegeben?
- Sind anfallende Versandkosten offensichtlich aufgeführt?
- Sind anfallende Lieferkosten offensichtlich aufgeführt?
- Sind Informationen zum Widerruf und zur Rückgabefrist offensichtlich angegeben?
- Falls Waren mit Batterien verkauft werden: Gibt es einen Hinweis über die Rücknahme und Entsorgung gebrauchter Batterien und Akkumulatoren (gemäß BattV – deutsche Batterieverordnung)?

Datenschutz
- Gibt es eine Datenschutzerklärung?
- Ist die Datenschutzerklärung auf einer separaten, eigens dafür angelegten Webseite des Webauftritts erreichbar?
- Gibt es Hinweise zur Speicherung und Löschung von personenbezogenen Daten?
- Wird der Benutzer vom Setzen eines Cookies unterrichtet (s. Cookie-Test, Kap. 9.6)?
- Ist der Inhalt dieser Unterrichtung bzgl. Cookies jederzeit abrufbar?
- Gilt für jedes eingesetzte Webanalyse-Tool:
  - Es ist in der Datenschutzerklärung aufgeführt?
  - Seine Arbeitsweise wird dort erläutert?
  - Die IP-Adressen werden anonymisiert?
  - Der Benutzer kann die Analyse wirksam unterbinden?
  - Es gibt einen Link, i. d. R. zum Tool-Hersteller, der zu weiteren Informationen führt?
- Gilt für jedes auf der Website eingebundene soziale Netzwerk:
  - Es ist in der Datenschutzerklärung aufgeführt?
  - Der Benutzer muss aktiv die Verbindung zum sozialen Netzwerk herstellen, zum Beispiel mit dem „2-Klick-Button“-Verfahren (s. o.)?
  - Der Benutzer kann die Verbindung zum sozialen Netzwerk wieder deaktivieren?
  - Es gibt einen Link zum sozialen Netzwerk, der zu weiteren Informationen bzw. zur dortigen Datenschutzerklärung führt?
- Wird die Arbeitsweise der „Gefällt-mir-Buttons“ erläutert?

Abb. 10.2 Beispiel Ghostery

Haftungsklauseln

Zwei Beispiele für Angaben auf der Website sind die Download-Absicherung und die Abstandsklausel zu externen Links:

1. **Beispiel – Download-Absicherung**:
„Das Herunterladen der Dateien erfolgt auf eigene Gefahr. Wir übernehmen keinerlei Haftung für Schäden, die direkt oder indirekt durch die Benutzung dieser Dateien entstehen. Dies gilt insbesondere dann, wenn diese Dateien für strafbare Handlungen eingesetzt wurden."
2. **Beispiel – Abstandsklausel**:
„Haftungshinweis: Trotz sorgfältiger inhaltlicher Kontrolle übernehmen wir keine Haftung für die Inhalte externer Links. Für den Inhalt der verlinkten Webseiten sind ausschließlich die Betreiber der verlinkten Webseiten verantwortlich."

► Eine Abstandsklausel schützt nicht vor Haftungsansprüchen! Daher müssen die Links vor der Veröffentlichung geprüft und diese Prüfung dokumentiert werden, mit Prüfer, Prüfdatum und Hardcopy.

Die in den Checklisten aufgeführten Inhalte können nicht alle möglichen, realen Situationen abdecken und erheben keinen Anspruch auf rechtliche Vollständigkeit und Richtigkeit.

10.1.6 Werkzeuge zum Content-Test

Im Internet gibt es jede Menge kostenlose Assistenten und Generatoren, die bei der Erstellung eines gesetzeskonformen Webimpressums helfen. Ein Disclaimer-Muster (Haftungsausschluss) und einen **Impressum-Generator** bietet zum Beispiel [URL: JuraForum] an.

GhosteryTM von Evidon [URL: Ghostery] zeigt in einer Liste alle auf einer Webseite eines Webauftritts eingebundenen Analysetools und sozialen Netzwerke an, wie z. B. in Abb. 10.2. Damit kann die Qualitätssicherung einfach überprüfen, ob die Angaben in der Datenschutzerklärung vollständig sind. Ghostery gibt es als Plugin für viele Browsertypen und als Web-App für das Mobile-iOS (iPhone, iPad). Die eingebundenen Verfahren lassen sich mit Ghostery auch blockieren.

10.1.7 Qualitätsanforderungen zum Content-Test

Die Qualitätsanforderungen an den Content einer Website werden durch den Erfüllungsgrad der entsprechenden Checklisten definiert.

Für den Nachweis der Rechtskonformität muss eine 100-prozentige Überdeckung der rechtlichen Fragen erreicht werden. D. h. 100 % der Fragen der Checklisten „Aufklärungspflicht" und „Webrecht" werden positiv beantwortet bzw. von einem juristisch ausgebildeten Prüfer als nicht relevant begründet.

Die Checkliste „Web-Content" muss nicht so restriktiv gehandhabt werden. Zum Beispiel kann mit einer Überdeckung von 90 % der Fragen die geforderte Qualität erreicht sein.

10.1.8 Empfehlungen zum Content-Test

Rechtsexperten einschalten
Die Fragen zu Webgesetzen und zum Webrecht sind nicht immer einfach und nicht ohne rechtliches Hintergrundwissen zu beantworten. Daher muss die Durchführung des Content-Tests für diese Themen durch Rechtsexperten erfolgen. Da sich die Rechtsgrundlage permanent ändern kann, ist die Aktualität der Checklisten zu gewährleisten. Fachleute müssen sie regelmäßig überprüfen und neu „justieren".

Prototyp prüfen
Der Content-Test sollte frühzeitig durchgeführt werden, z. B. wenn die Website als Design-Entwurf oder besser als Prototyp vorliegt, denn die Ergebnisse der Prüfung können entscheidenden Einfluss auf das Design haben.

Finaler Content-Test zur Freigabe
Vor der Produktivstellung der Website muss abschließend sichergestellt werden, dass die im Content-Test geprüften Inhalte nachträglich nicht verändert wurden. Daher empfiehlt sich ein wiederholter Check des Contents vor der Freigabe der Anwendung.

10.2 Oberflächentest

Eine Anwendung sollte sich dem Benutzer in allen Bereichen gleich darstellen. Standardfunktionalitäten wie das Navigieren und das Drucken müssen einheitlich sein und funktionieren. Diese Anforderungen an eine Software werden i. d. R nicht explizit vom Auftraggeber spezifiziert, sondern als selbstverständlich vorausgesetzt. Zu prüfen sind sie dennoch im Oberflächentest.

► Der **Oberflächentest** überprüft die Einhaltung von Dialogrichtlinien und die Korrektheit von Standardfunktionalitäten auf der Benutzeroberfläche einer Anwendung.

10.2.1 Dialogrichtlinien und Standardfunktionalitäten

Der Oberflächentest stellt sicher, dass sich eine Website oder eine Applikation bei unterschiedlichen technischen Voraussetzungen korrekt darstellt und der Nutzer zielgerichtet und fehlerfrei durch die Anwendung navigieren kann. Zudem werden Design-Standards überprüft, die für jede Anwendung in einer Organisation gelten sollten und unabhängig von der fachlichen Funktionalität sind.

Diese Trennung von den fachlichen Funktionstests hat zwei Vorteile. Die Testfälle sind auf andere Anwendungen – zumindest innerhalb eines Unternehmens – übertragbar und der Oberflächentest kann von Personen durchgeführt werden, die sich nicht in die fachlichen Funktionen der Anwendung einarbeiten müssen.

Der Oberflächentest ist ein dynamischer Test und nicht mit dem statischen Content-Test (s. Kap. 10.1) gleichzusetzen, bei dem die Vollständigkeit und die Konformität der darzustellenden Web-Inhalte ohne Ausführung von Testfällen geprüft werden.

Die Testfälle zum Oberflächentest werden anhand der Fragen der folgenden Checkliste entworfen, welche die Forderungen der ISO 9241 Teil 110, Grundsätze der Dialoggestaltung (s. Abschn. 2.2.1), berücksichtigt und zudem Prüfpunkte zu erfahrungsbasierten Fehlerquellen enthält:

✓ Checkliste Oberflächentest

Aufgabenangemessenheit

- Sind alle für die Ausführung einer Aufgabe benötigten Menüpunkte sichtbar und aktiviert?
- Sind alle für die Ausführung einer Aufgabe nicht benötigten Menüpunkte deaktiviert oder nicht sichtbar?
- Werden keine aufgabenfremden, unnötigen Eingaben gefordert (z. B. irrelevante, persönliche Angaben zu einer Bestellung)?
- Werden nicht benötigte Eingabefelder kontextsensitiv ausgeblendet?
- Werden Eingabeobjekte je nach Anforderung korrekt gesperrt bzw. entsperrt?

Erwartungskonformität

- Erscheinen alle wiederkehrenden Elemente auf jeder Seite an derselben Stelle in derselben Darstellung (Firmenlogo, Link zum Impressum, Steuerelemente, …)?
- Führt das Firmenlogo stets zur Homepage?
- Haben alle gleichen Elemente dieselbe Darstellung (Farbe und Größe von Buttons, Menüpunkte, Textformate, …)
- Werden Texte und Oberflächenelemente auf allen Seiten gleich dargestellt?
- Sind Begriffe auf allen Seiten identisch, z. B. ist der „Warenkorb“ immer und in allen Zusammenhängen der „Warenkorb“?
- Sind unterstrichene Wörter stets Links?

- Werden gängige Icons verwendet und sind sie aussagekräftig in Bezug auf ihre Verwendung?
- Sind Icons mit erklärenden Texten versehen?

Fehlertoleranz
- Werden Plausibilitätsprüfungen so bald wie möglich durchgeführt?
- Werden Eingaben möglichst vor dem Abschicken einer Seite überprüft?
- Steht der Cursor bei falscher Eingabe auf dem entsprechenden Feld?
- Ist die Prüfreihenfolge der Eingaben logisch und korrekt (Felder, Feldkombinationen)?
- Erfolgt die Fehlerprüfung in allen Windows/Dialogen gleich, entweder fensterweise oder eingabefeldweise?

Selbstbeschreibungsfähigkeit[3]
- Weiß der Benutzer zu jedem Zeitpunkt, an welcher Stelle er sich befindet (z. B. durch Breadcrumbs[GL])?
- Wird der Status der Bearbeitung stets am unteren oder oberen Bildrand korrekt angezeigt?
- Wird nach einer Suche die Anzahl der Treffer angezeigt?
- Wird bei Verarbeitungen, die von kurzer Dauer sind, stets die Sanduhr angezeigt und bei länger dauernden ein Fortschrittsbalken oder ein anderes animiertes Symbol?
- Sind die Pflichtfelder von den optionalen Feldern optisch zu unterscheiden?
- Werden besuchte Links als solche farblich dargestellt?

Steuerbarkeit
- Können Tabellen spaltenweise sortiert werden?
- Kann für Suchvorgänge die Anzahl der anzuzeigenden Treffer pro Seite festgelegt werden?
- Gibt es Filtermöglichkeiten für große Treffermengen?
- Können Downloads unterbrochen werden?
- Kann das „Intro“ auf der Homepage übersprungen werden?[4]
- Sind Video- und Audiosequenzen abschaltbar?
- Gibt es bei zeitintensiven Verarbeitungen eine Abbruchmöglichkeit und funktioniert dieser Abbruch?
- Können umfangreiche Eingaben gespeichert und wieder aufgenommen werden?
- Werden schnell ladbare Alternativen für komplexe Seiten und Inhalte angeboten?
- Kann automatisch ein Bookmark auf die Homepage gesetzt werden?
- Kann von jeder Seite auf die Homepage gesprungen werden.

[3] Beispiele und Abbildungen zur Selbstbeschreibungsfähigkeit und Steuerbarkeit s. Abschn. 2.2.1

[4] Persönlicher Aufruf: Weg mit nervigen Intro-Videos!

Navigation

- Sind Navigationselemente (Buttons, Menüpunkte) eindeutig, übersichtlich, verständlich und auffindbar?
- Entsprechen die Navigationselemente der inhaltlichen Gliederung (verzweigt z. B. jeder Menüpunkt zum richtigen Inhalt)?
- Funktionieren alle Navigationselemente (Buttons, Menüpunkte, Hyperlinks) richtig?
- Sind Navigations- und Präsentationsbereich getrennt?
- Ist die Navigationsleiste auf jeder Seite sichtbar und einheitlich?
- Existiert eine vollständige und funktionstüchtige Sitemap[GL]?
- Ist die Navigation der App von der Browser-Navigation unabhängig?
- Werden horizontale Scroll-Balken vermieden?
- Steht in der Titelleiste des Browsers immer der Name der aktuellen Seite?
- Kann der Benutzer über die Breadcrumb-Leiste[GL] navigieren?
- Kann über die Browser-Navigation vor- und zurückgeblättert werden, ohne dass Datenverluste oder Orientierungsprobleme entstehen?
- Existiert auf jeder Webseite ein Link zur
  - Nächst höheren Hierarchiestufe?
  - Hilfefunktion?
  - Homepage?
- Ist jede Funktion mit Tabulator-Tasten erreichbar?
- Erfolgt die Cursor-Steuerung mit den Tasten <TAB> und <SHIFT+TAB> in der richtigen Reihenfolge?
- Werden Shortcuts (Tastaturkurzbefehle) für wichtige Funktionen bereitgestellt und funktionieren diese?

Darstellbarkeit

- Werden die Seiten bei allen möglichen Bildschirmauflösungen korrekt und vollständig angezeigt?
- Sind die Inhalte insbesondere auch auf Bildschirmen mit kleiner Auflösung nutzbar?
- Ist die Anwendung bei allen möglichen Bildschirmauflösungen funktionstüchtig?
- Ist der Kontrast von Texten und Grafiken hinreichend für eine Schwarz/Weiß-Darstellung?
- Werden durchgängig Cascading Style Sheets[GL] (CSS) zur Darstellung der Webinhalte verwendet, sofern kein Content-Management-Systemen eingesetzt wird?
- Sind Lesbarkeit und Handhabbarkeit bei Veränderung der Schriftart und Schriftgröße in den Browser-Einstellungen gewährleistet?
- Gibt es deutlich lesbare, alternative Texte für grafische Navigationselemente?
- Sind alle Links mit einer eindeutigen Beschreibung des Zielobjektes versehen?
- Sind Links in Grafiken deutlich erkennbar?
- Sind ungewöhnliche Links mit Hinweisen versehen (z. B. fremde Sprache, extreme Dateigröße, ungewöhnliches Format)?

Initialisierung
- Sind beim Start der Anwendung/einer Seite alle Default-Werte richtig gesetzt?
- Sind alle Ausrichtungen korrekt?
- Sind Sortierungen richtig?
- Sind alle Eingabefelder sinnvoll vorbelegt (z. B. Datumsfelder)?
- Sind Radiobuttons richtig initialisiert?
- Werden Bilder und Grafiken mit einer Vorschaugrafik geladen?

Standardfunktionalitäten
- Funktionieren die Suchfunktionen korrekt?
- Funktionieren die Sortierfunktionen korrekt?
- Arbeitet die Funktion „Zurücksetzen“ stets richtig?
- Läuft die Guided Tour korrekt ab?
- Arbeiten die Druckfunktionen des Browsers mit der Anwendung korrekt zusammen?
- Arbeiten die Druckfunktionen innerhalb der Anwendung korrekt?
- Gibt es für den Druck der Webseiten ein extra CSS-Druck-Layout?
- Gibt es eine Druckvorschau?
- Können alle Inhalte einwandfrei gedruckt werden?
- Gehen keine Inhalte verloren, wenn Seiten für den Hochkant-Ausdruck vorgesehen sind?
- Gehen keine Inhalte verloren, wenn Seiten für den Quer-Ausdruck vorgesehen sind?

An dieser Stelle ließe sich sicherlich deftig streiten, ob zum Beispiel die in der Checkliste „Oberflächentest“ bei den Themen „Initialisierung“ und „Standardfunktionalitäten“ aufgeführten Prüfpunkte besser zum Kapitel „Tests zur Funktionalität“ (s. Kap. 9) gehören. Kann man machen. Aber weil diese Punkte „selbstverständliche“ Eigenschaften einer Anwendung prüfen, die i. d. R. nicht explizit angefordert werden, sind sie hier dem Benutzbarkeitstest zugeordnet. Nichterfüllung schlägt sich nicht unbedingt in fachlichen oder technischen Fehlern nieder, wohl aber auf die Zufriedenheit der Benutzer.

Bleiben noch Besonderheiten zum Oberflächentest für Mobile-Websites mit der Checkliste „Mobile-Oberflächentest“ zu ergänzen.

✓ Checkliste Mobile-Oberflächentest
Navigation
- Gibt es durchgängig eine funktionierende Schaltfläche „Zurück“?
- Gibt es durchgängig eine funktionierende Schaltfläche „Startseite“ bzw. „Homepage“?
- Wird horizontales Scrollen vermieden?
- Gibt es keine Rollover-Effekte[GL]?

- Ist das Suchfeld an prominenter Stelle positioniert?
- Kann der Nutzer einfach zwischen der Desktop-Website und der Mobile-Website wechseln?

Darstellbarkeit
- Ist die Mobile-Website leicht überschaubar?
- Ist der Inhalt gut lesbar, ohne dass gezoomt werden muss?
- Kann der Benutzer die Schriftgröße ändern?
- Sind Schaltflächen mit 3D-Effekten oder Schattierungen gut erkennbar dargestellt?
- Werden die Webseiten im Hoch- und Querformat korrekt angezeigt?
- Passt sich die Website der Größe und der Auflösung des Endgerätes an?
- Werden Bilder und Grafiken in kleinskalierter Vorschau angezeigt?
- Sind alle Bilder und Grafiken so komprimiert, dass sie schnell geladen werden?
- Gibt es keine Flash-Animationen?

Erwartungskonformität
- Sind die wichtigsten Informationen der Desktop-Website verfügbar?
- Sind die wichtigsten Funktionen der Desktop-Website verfügbar?
- Können Suchanfragen und Produktwünsche gespeichert werden?

Handhabbarkeit
- Können Eingaben mit allen unterstützten Formaten erfasst werden:
  - Interne Tastatur?
  - Externe Tastatur?
  - Verschiedene Tastatur-Layouts (QWERTY, QWERTZ, 12er, …)?
  - Interner und externer Touchscreen?
  - Eingabestift?
- Sind die Schaltflächen und Links hinreichend groß und gut sichtbar?
- Sind alle Schaltflächen und Links mit einem Finger ausführbar?
- Ist er Abstand zwischen den Schaltflächen und Links so groß, dass versehentliche Klicks vermieden werden?
- Sind Bereiche um kleinere Schaltflächen herum auch anklickbar und lösen das richtige Ereignis aus?[5]
- Sind Checkboxen auch über den beschreibenden Text anklickbar und nicht nur innerhalb der Checkbox?
- Ist die Anzahl der Links auf den einzelnen Webseiten minimiert?

[5] „Dicke-Finger-Bedienbarkeit“ ist ein geeigneter Kandidat für ein neues Teilqualitätsmerkmal der Benutzbarkeit

Steuerbarkeit
- Kann der Nutzer für spätere Aufrufe der Website seine bevorzugte Version der Website (Mobile, Desktop, …) festlegen?

10.2.2 Stichprobentest der Oberfläche

Um den Aufwand eines Cross-Browser-Tests (s. Abschn. 9.11.1) zu reduzieren, wird der komplette Oberflächentest nur für die am häufigsten eingesetzten Browser-Versionen, wie im vorherigen Abschnitt beschrieben, durchgeführt. Für die seltener installierten Browser-Versionen werden aus der Checkliste zum Oberflächentest die wichtigsten Testfälle ausgewählt und in Stichproben geprüft.

✓ Checkliste Stichprobe Browser-Test
Allgemeine Darstellung
- Sind Lesbarkeit und Handhabbarkeit bei Veränderung der Schriftart und Schriftgröße in den Browser-Einstellungen gewährleistet?
- Haben alle gleichen Elemente dieselbe Darstellung (Farbe und Größe von Buttons, Menüpunkte, Textformate, Breite von Listboxen, …)?
- Werden alle Elemente im Browser richtig dargestellt?
- Wird die Anwendung bei allen Bildschirmauflösungen richtig dargestellt?
- Werden die verwendeten Cascading Style Sheets richtig interpretiert?

Navigation
- Sind alle Navigationselemente (Buttons, Menüpunkte, Hyperlinks) auffindbar?
- Funktionieren alle Navigationselemente (Buttons, Menüpunkte, Hyperlinks) richtig?
- Werden die alternativen Texte für grafische Navigationselemente angezeigt?
- Ist die Navigationsleiste auf jeder Seite sichtbar?
- Erfolgt die Cursor-Steuerung mit <TAB> und <SHIFT+TAB> in der richtigen Reihenfolge?

Druck
- Arbeitet die Druckfunktion des Browsers korrekt?
- Arbeiten die Druckfunktionen innerhalb der Anwendung korrekt?

Was dem Desktop-Browser recht ist, ist dem Mobile-Browser billig. Hier die Checkliste „Stichprobe Mobile-Browser-Test“ für die wichtigsten Checks:

✓ Checkliste Stichprobe Mobile-Browser-Test
Navigation
- Gibt es keine Rollover-Effekte[GL]?
- Kann der Nutzer einfach zwischen der Desktop-Website und der Mobile-Website wechseln?

Darstellbarkeit
- Ist der Inhalt der mobilen Website gut lesbar, ohne dass gezoomt werden muss?
- Werden die Webseiten im Hoch- und Querformat korrekt angezeigt?

Handhabbarkeit
- Sind Schaltflächen und Links mit einem Finger bedienbar?
- Sind die Schaltflächen und Links hinreichend groß und gut sichtbar?
- Sind Bereiche um kleinere Schaltflächen herum auch anklickbar und lösen das richtige Ereignis aus?
- Können Eingaben mit allen unterstützten Formaten erfasst werden?

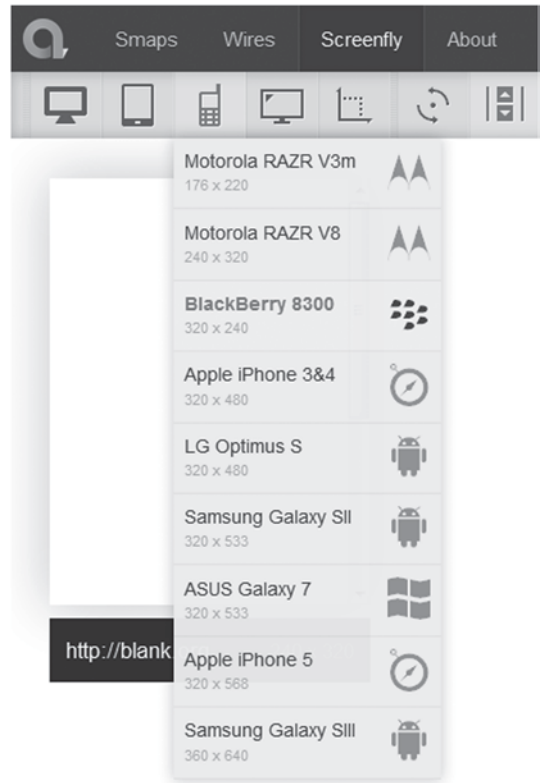
10.2.3 Werkzeuge für den Oberflächentest

Wenn es darum geht, die Anpassungsfähigkeit einer Webseite an Größe und Auflösung des Endgerätes zu überprüfen (**Adaptivität**), eignet sich dafür neben der Web Accessibility Toolbar und Web Developer Toolbar (s. Abschn. 9.8.2) auch das kostenlose Online-Tool **Screenfly** [URL: Screenfly]. Damit können spezielle Tablet-PC oder Mobile-Geräte für den Test ausgewählt werden (s. Abb. 10.3).

Der Oberflächentest kann anhand der Checkliste „Oberflächentest“ manuell durchgeführt werden. Mit einem **Capture Replay Tool** (s. Kap. 14.3) können einige der Checkpunkte zum Oberflächentest automatisch überprüft werden, denn ein solches Werkzeug kann zum Beispiel die Einhaltung vorgegebener Standards (Farbgebung, Schriftart, …), die Existenz von Objekten (Buttons, Menüpunkte, Navigationshilfen, …) und die Eigenschaften von Objekten (Größe, Position, …) erkennen.

Wenn dem Projekt ein Capture Replay Tool zur Verfügung steht und die vereinbarten Standards für mehrere Webauftritte oder Apps getestet werden müssen, lohnt sich der Aufwand, standardisierte Skripts zur Oberflächenprüfung zu entwickeln und die Tests zu automatisieren. Dafür werden allerdings Testspezialisten benötigt. Im Testkonzept ist festzulegen, welche Prüfungen automatisiert werden sollen. Die entsprechenden Testfälle werden dann in der Checkliste zum Oberflächentest gestrichen. Folgende Punkte sind u. a. geeignete Kandidaten, die mit einem Capture Replay Tool automatisch überprüft werden können:

Abb. 10.3 Screenfly



- Funktionieren alle Navigationselemente (Buttons, Menüpunkte, Hyperlinks) richtig?
- Ist die Navigationsleiste auf jeder Seite sichtbar?
- Gibt es alternative Texte für grafische Navigationselemente?
- Kann von jeder Seite die Homepage aufgerufen werden?
- Erscheinen alle wiederkehrenden Elemente auf jeder Seite an derselben Stelle in derselben Darstellung (Steuerelemente, Firmenlogo, Link zum Impressum, …)?
- Sind unterstrichene Wörter stets Links?
- Haben alle gleichen Elemente dieselbe Darstellung (Farbe und Größe von Buttons, Menüpunkte, Textformate, …)
- Werden Texte und Oberflächenelemente auf allen Seiten gleich dargestellt?
- Wird der Bearbeitungsstatus am unteren Bildrand korrekt angezeigt?
- Sind alle Default-Werte beim Laden einer Webseite/eines Formulars richtig gesetzt?
- Arbeitet die Funktion „Zurücksetzen“ stets korrekt?

10.2.4 Qualitätsanforderungen zum Oberflächentest

Die Qualitätsmerkmale Aufgabenangemessenheit, Erwartungskonformität, Fehlertoleranz, Lernförderlichkeit, Selbstbeschreibungsfähigkeit, Steuerbarkeit, Funktionalität und Benutzbarkeit sind in der Checkliste „Oberflächentest“ mit konkreten Prüfpunkten versehen, sodass die Qualitätsanforderungen an den Oberflächentest durch den vorzugebenden Erfüllungsgrad der Checkliste festgelegt werden.

10.2.5 Empfehlungen zum Oberflächentest

Der Oberflächentest kann von Testern ohne fachliche oder technische Spezialkenntnisse durchgeführt werden. Daher ist es sinnvoll, den Oberflächentest von den komplexeren Funktionstests, die nur von „teuren“ Spezialisten durchgeführt werden können, zu trennen.

Um den Aufwand des Oberflächentests gering zu halten, sollte dieser nur einmal komplett durchgeführt werden, nämlich mit dem Browser-Typ, den die Anwender am häufigsten einsetzen. Für die anderen Browser-Typen kann der Oberflächentest eingeschränkt durchgeführt werden.

Apple, BlackBerry, Google und Microsoft haben für Apps, die über die entsprechenden Stores vertrieben werden, Programmierrichtlinien für ihre mobilen Geräte festgelegt. Die Inhalte dieser Richtlinien liefern weitere, nicht nur gerätespezifische Prüfpunkte für die obige Checkliste „Oberflächentest-Mobile“. Auch wenn die zu testende App nicht über einen dieser Stores verteilt wird, kann sich ein Blick in die Guidelines für Entwickler und Tester lohnen.

10.3 Usability-Test

Wenn eine Webanwendung der gesamten Webgemeinde zur Verfügung gestellt wird, sind die Nutzer unbekannt. Sie sprechen verschiedene Sprachen, haben keine Schulung für die Anwendung der Software erhalten, haben kein Handbuch gelesen und treffen mit unterschiedlichen Intentionen auf die Website. Trotzdem sollen sich alle Benutzer in der Anwendung zurechtfinden und sie optimal für ihre Zwecke nutzen können. Das gilt natürlich auch für Web- und Mobile-Apps, die von unterschiedlichsten Nutzerkreisen eingesetzt werden können.

► Der **Usability-Test** überprüft in Abhängigkeit der zu erreichenden Zielgruppen die Gebrauchstauglichkeit (Usability)[6] einer Website oder Applikation.

[6] S. Abschn. 2.2.1, Normenbestimmte Qualitätsmerkmale

Das Problem beim Nachweis der Gebrauchstauglichkeit einer Anwendung ist, dass die Bewertung subjektiv ist. Als Konsequenz daraus sollten Repräsentanten aus unterschiedlichen Nutzergruppen die Anwendung in einem Usability-Labor ausprobieren und anschließend eine Bewertung abgeben.

Als Ergänzung oder Alternative zum Usability-Labor können Befragungen der einzelnen Nutzergruppen mit Fragebögen durchgeführt werden.

10.3.1 Usability-Labor

Für den Usability-Test müssen die Nutzergruppen der zu testenden Anwendung identifiziert und beschrieben werden. Das können zum Beispiel Gelegenheits-Surfer, Interessenten oder Kunden sein.

- Gelegenheits-Surfer stoßen mehr oder weniger zufällig auf die Website. Hier entscheiden die ersten Sekunden, ob der Besucher von ihren Inhalten gefesselt wird.
- Interessenten steuern, eventuell über eine Suchmaschine, die Website gezielt an, um für sie relevante Informationen zu erhalten.
- Kunden wollen Geschäfte über die Webapplikation abwickeln. Sie wollen ihre Transaktionen[GL] reibungslos, sicher und fehlerfrei durchführen.

Für jede Nutzergruppe werden konkrete Aufgaben festgelegt, die mit der Anwendung in einer vorgegebenen Zeit gelöst werden sollen, wie zum Beispiel die drei Aufgaben zu unserer Autofinanzierung:

1. Aufgabe: Die Testnutzer der Zielgruppe Kunden sollen in 15 min für ihr Wunschauto eine Finanzierungsberechnung durchführen und das Ergebnis ausdrucken.
2. Aufgabe: Interessenten sollen in 6 min herausfinden, welche unterschiedlichen Finanzierungsmöglichkeiten angeboten werden.
3. Aufgabe 3: Informationssuchende sollen innerhalb von 2 min allgemeine Informationsunterlagen für eine Autofinanzierung anfordern, die auf dem Postweg zugesendet werden.

Zu jeder Nutzergruppe werden mehrere neutrale Personen ausgewählt und aufgefordert, die gestellten Aufgaben in einem Usability-Labor zu lösen. Während der Sitzung werden die Testnutzer beobachtet und ihre Aktionen werden eventuell mit einer Kamera aufgezeichnet. Klickpfade und Kennzahlen, wie benötigte Zeit pro Aufgabe, Anzahl der Klicks pro Aufgabe oder Verweildauer auf einem Dokument, werden protokolliert. Die Kommentare der Testnutzer werden ebenfalls festgehalten.

10.3.2 Befragung

Nach der Sitzung im Usability-Labor werden die Testnutzer nach Ihren persönlichen Erfahrungen und Eindrücken befragt. Die Befragung kann durch einen Fragebogen und alternativ oder ergänzend in einem Interview erfolgen. Sie besteht aus zwei Teilen.

Zuerst werden die Vorkenntnisse und das allgemeine Nutzungsverhalten erfragt, um die Benutzer in ihren Bewertungen einordnen zu können.

1. Fragen zum Nutzungsverhalten

Allgemeines

- Wie viele Stunden pro Woche nutzen Sie das Internet (privat/dienstlich)?
- Welchen technischen Zugang haben Sie zum Internet (DSL, Glasfaser, Satellit, VDSL, …)?
- Haben Sie Programmierkenntnisse? Wenn ja, welche?

Desktop-PC

- Welches Betriebssystem benutzen Sie?
- Welche(n) Browser benutzen Sie?
- Welche Browser-Einstellungen haben Sie normalerweise aktiviert? (JavaScript, Java, Cookies, Warnmeldungen bei Cookies, Blocken von PopUp-Fenstern, …)
- Mit welchen Bildschirmauflösungen arbeiten Sie?

Mobile-Geräte

- Nutzen Sie ein Smartphone oder einen Tablet-PC mit Internet-Zugang?
- Nutzen sie mehrere mobile Endgeräte?
- Welches mobile Gerät setzen Sie am häufigsten ein (Hersteller, Typbezeichnung)?
- Welchen Provider haben Sie?

2. Bewertung durch Benutzer

Im zweiten Teil einer Befragung füllen die Testnutzer einen Bewertungsbogen zur Benutzerführung, zur Struktur, zum Design und zum Inhalt der Anwendung aus. Das tun sie auf Basis ihrer persönlichen Erfahrungen, die sie beim Lösen der gestellten Aufgaben mit der Anwendung gemacht haben.

Das Bewertungsmaß ist auf einer Bewertungsskala mit einer geraden Anzahl von Antwortmöglichkeiten, die eine positive oder negative Entscheidung zu jeder Frage erzwingt, vorgegeben. Am Ende eines Fragebogens oder Interviews werden noch Verbesserungsvorschläge von den Testnutzern eingefordert.

Ein Bewertungsbogen für eine Web-App könnte folgende Inhalte haben:

Merkmal	Zu bewertende Aussage	++	+	-	-
Aufgabenangemessen	Ich hatte nie das Gefühl, mit meiner Aufgabe nicht weiter zu kommen				
	Ich bin nie mit Informationen überfrachtet worden				
Erwartungskonform	Die Seitengestaltung entspricht meinen Vorstellungen				
	Ich konnte die Schrift gut lesen				
Fehlertolerant	Hinweise und Fehlermeldungen sind eindeutig und verständlich				
Lernförderlich	Die Web-Applikation ist einfach zu erlernen				
	Die Beispiele zu jeder Teilfunktion haben mir geholfen, die Vorgänge zu verstehen und nachzuvollziehen				
	Es fiel mir bei keiner Funktion schwer, sie zu wiederholen				
Selbstbeschreibungsfähig	Die Texte und Beschreibungen sind eindeutig und verständlich				
	Ich habe alle Menüpunkte schnell gefunden				
	Ich wusste immer, wo ich mich befand				
	Die Beschreibungstexte der Links beschreiben eindeutig das jeweilige Ziel				
	Ich war auf die Hilfefunktion nicht angewiesen				
Steuerbar	Die Navigation ist klar und übersichtlich				
	Ich wusste immer, wie ich zu einer bestimmten Funktion komme				
	Die benötigten Links habe ich immer schnell gefunden				
	Ich konnte mich an jeder Stelle einfach entscheiden, wie ich weiter machen sollte				
	Ich wusste immer, wie ich zu einem bestimmten Punkt gekommen war				
	Ich hatte nie Schwierigkeiten, den für die Aufgabe optimalen Weg zu finden				
Meine Verbesserungsvorschläge:					

10.3.3 Auswertung Usability-Labor und Befragungen

Die Auswertungen der Aufzeichnungen aus dem Usability-Labor, der Fragebögen und der Interviews geben Aufschluss über die Stärken und Schwächen der Webanwendung, weil Fragen quantitativ beantwortet werden können. Die Usability wird somit messbar.

Allgemeine Aussagen

- Wie viel Prozent der Teilnehmer haben Ihre Aufgabe erfüllt?
- Wie viel Prozent der Teilnehmer haben aufgegeben?
- Wie oft wurde die Hilfefunktion aufgerufen?
- Warum wurde die Hilfefunktion bei Problemen nicht aufgerufen?
- Haben die Personen zur Lösung der Aufgabe immer den kürzesten Pfad durch die Anwendung genommen?
- Wie lange haben die Personen für die Erledigung ihrer Aufgabe gebraucht?
- An welchen Dokumenten und Funktionen haben sich die Personen relativ lange aufgehalten?

Aufgabenspezifische Aussagen
hier am Beispiel der im Abschn. 10.3.1 gestellten Aufgaben:

Zur 1. Aufgabe:
- Waren die Eingaben für eine Finanzierungsberechnung auf Anhieb fehlerfrei?
- Konnten alle Personen die Finanzierungsberechnung drucken?

Zur 2. Aufgabe:
- Konnten die Testnutzer alle Finanzierungsvarianten aufzählen?
- Konnten die Testnutzer alle Finanzierungsvarianten erklären?

Zur 3. Aufgabe:
- Wurde die Funktion zur Anforderung der Finanzierungsunterlagen für den Postweg ohne Umwege gefunden?
- Waren die Eingaben zur Anforderung der Finanzierungsunterlagen für den Postweg fehlerfrei?

Auswertungen
Wenn die Mehrzahl der Testnutzer die ihnen gestellte Aufgabe nicht, mit Problemen oder nur in unverhältnismäßig langer Zeit lösen konnte, müssen Effektivität und Effizienz der Gebrauchstauglichkeit verbessert werden.

Der Grad der Zufriedenstellung der Benutzer ergibt sich aus den Antworten des ausgefüllten Fragebogens und der geführten Interviews. Haben sich die Benutzer darin zu den gleichen Themen negativ geäußert oder gleichartige Verbesserungsvorschläge gemacht, besteht Verbesserungsbedarf.

10.3.4 Blickregistrierung

Ein ergänzendes Verfahren zum Usability-Test ist die **Blickregistrierung (Eye Tracking)**, bei der die Blickbewegungen eines Testnutzers aufgezeichnet und ausgewertet werden.

Die Blickregistrierung zur Analyse des Benutzerverhaltens auf einer Website zeigt zum Beispiel auf,

- was Benutzer auf einer Bildschirmseite wahrnehmen,
- wie oft sie welche Bereiche mit ihren Blicken erfassen,
- wie lange und intensiv sie ein einzelnes Element (Seite, Menüpunkt, Überschrift, Grafik, Text, …) betrachten und
- ob sie bestimmte Elemente besonders häufig oder gar nicht wahrnehmen.

Zur Blickregistrierung werden spezielle Werkzeuge, sogenannte Eye Tracker benötigt, wie im folgenden Abschnitt beschrieben.

10.3.5 Werkzeuge für den Usability-Test

Das Benutzerverhalten der Testnutzer wird mittels **User Tracking Tools** (alias Webanalyse-Tools) protokolliert, um zum Beispiel zu folgenden Fragen Auskunft zu bekommen:

- An welchen Tagen und zu welcher Uhrzeit waren wie viele Besucher auf der Website?
- Welche Seiten wurden am häufigsten und mit welcher durchschnittlichen Dauer besucht?
- Wann hat wer wie lange mit welcher (anonymisierten!) IP-Adresse über welchen Provider die Website besucht?
- Wie lange war ein Benutzer auf welcher Seite?
- Welchen **Klickpfad** hat der Benutzer genommen? (Auf welcher Seite ist der Benutzer eingestiegen, welchen Pfad hat er durch die Anwendung genommen und auf welcher Seite ist er wieder ausgestiegen?)
- Mit welchen Suchwörtern gelangen Benutzer über Suchmaschinen auf die Website?

Diese Aufzählung ist nur eine Auswahl der Auswertungsmöglichkeiten, die User Tracking Tools bieten. User Tracking Tools werden nicht nur im Usability-Labor im Rahmen des Usability-Tests eingesetzt, sondern auch zur Erhebung von Statistiken während des laufenden Betriebs. Eine Übersicht von User Tracking Tools liefert [URL: UserTrackingTools].

Seite	Seitenaufrufe	Eindeutige Seitenaufrufe	Durchschn. Besuchszeit auf Seite
	3.125 % des Gesamtwerts: 100,00 % (3.125)	2.759 % des Gesamtwerts: 100,00 % (2.759)	00:00:41 Website-Durchschnitt: 00:00:41 (0,00 %)
1. /	669	574	00:00:37
2. /[redacted].html	466	450	00:00:45
3. /[redacted].html	137	106	00:00:23
4. /[redacted]html	136	135	00:01:36
5. /[redacted].html	103	87	00:00:40
6. /impressum.html	86	76	00:00:31
7. /[redacted].html	84	77	00:01:46

Abb. 10.4 Statistik Seitenaufrufe

Die mit Google Analytics erzeugte Statistik in Abb. 10.4 zeigt beispielhaft die Anzahl der Seitenaufrufe und die durchschnittliche Verweildauer der Besucher auf den einzelnen Webseiten.

► Achtung: Die korrekte Verwendung dieser Analysewerkzeuge muss in der Datenschutzerklärung dokumentiert sein (s. Abschn. 10.1.5, Checkliste „Rechtliche Angaben")!

Für die **Blickregistrierung** werden **Eye Tracker** benötigt, deren Einsatz aufgrund der technischen Anforderungen sehr kostspielig ist. Sie sind im Web unter den Stichworten „Eye Tracker" oder „Blickregistrierung" zu finden. Eye Tracker messen zum Beispiel:

- Anzahl der Fixationen zur Durchführung einer bestimmten Aufgabe in einem bestimmten Zeitintervall
- Anzahl der Fixationen eines Objektes auf der Webseite im Verhältnis zur Anzahl aller Fixationen auf der Webseite in einem bestimmten Zeitintervall in Prozent
- Länge des Blickpfades (Summe der Abstände zwischen den Orten der einzelnen Fixationen) zur Durchführung einer bestimmten Aufgabe
- Anzahl der Sakkaden zur Durchführung einer bestimmten Aufgabe in einem bestimmten Zeitintervall

Eine große Anzahl der Sakkaden kann bedeuten, dass der Benutzer viele Informationen auf der Webseite „zusammensuchen“ muss. Bleiben noch die Begriffe Fixation und Sakkade zu definieren (Zitat aus [URL: wiki-Blick]):

> Fixationen und Sakkaden machen den größten Teil der bewussten Augenbewegungen aus. Während einer **Fixation** nimmt das Auge über die Netzhaut Informationen aus der Umgebung auf und leitet diese nach einer Vorverarbeitung an das Gehirn weiter. Während einer **Sakkade** hingegen nimmt das Auge keine visuellen Informationen auf. Man ist in dieser Phase tatsächlich blind und sieht darin eine der Mitursachen der Unaufmerksamkeitsblindheit, also der Unempfänglichkeit für visuelle Reize durch mangelnde Aufmerksamkeit.

10.3.6 Online-Umfrage

Wenn die Einrichtung eines Usability-Labors zu aufwändig oder zu teuer ist, bietet sich eine Pilotphase an, in der dem Nutzer **Online-Umfragen** zur Verfügung gestellt werden. Anreize für Nutzer, einen Fragebogen online auszufüllen, können auf der Website zum Beispiel durch besondere Angebote oder ein Preisausschreiben gegeben werden.

Der Online-Fragebogen enthält Fragen zum allgemeinen Nutzungsverhalten und zur Bewertung der Gebrauchstauglichkeit, s. Abschn. 10.3.2.

10.3.7 Qualitätsanforderungen zum Usability-Test

Qualitätsanforderungen zu den Qualitätsmerkmalen Gebrauchstauglichkeit und Benutzerfreundlichkeit müssen nicht nur für die Auswertungen im Usability-Labor festgelegt werden, sondern auch für die Zeit nach der Freigabe der Anwendung. Das setzt voraus, dass das Benutzerverhalten auch nach der Einführung der Software aufgezeichnet und ausgewertet wird. Qualitätsanforderungen zum Usability-Test können wie folgt definiert werden:

Qualitätsanforderungen an das Usability-Labor

- Die Testnutzer haben zu 90 % die ihnen gestellte Aufgabe in der vorgegebenen Zeit gelöst.
- 75 % der Testnutzer haben sich positiv geäußert.
- Jede Frage aus dem Bewertungsfragebogen wurde von mindestens 65 % der Befragten positiv („++“ oder „+“) bewertet.

Qualitätsanforderungen an den laufenden Betrieb

- 30 % der Benutzer verweilen durchschnittlich mindestens 2 min auf der Website.
- Maximal 3 % der Benutzer, die länger als 5 min auf der Website verweilen, stellen Supportanfragen.

- Die Anzahl der Anforderungen von Informationsmaterial steigt im 1. Jahr um 20 % pro Monat.
- Die Anzahl der Vertragsabschlüsse steigt im 1. Jahr um 10 % pro Monat.

10.3.8 Empfehlungen zum Usability-Test

Der Usability-Test sollte kurz vor der Freigabe der Anwendung im Usability-Labor durchgeführt werden. Die Testnutzer sollten repräsentativ für jede Zielgruppe ausgewählt und nicht aus dem Projektumfeld rekrutiert werden, damit die Testergebnisse authentisch sind.

Die Planung, Durchführung und Auswertung des Usability-Tests sollte von Fachleuten aus dem Marketingbereich vorgenommen werden. Wenn der geschäftliche Erfolg des Unternehmens von der Anwendung abhängt, ist es ratsam, den Usability-Test von einem externen Spezialisten durchführen zu lassen.

Die Auswertungen des Benutzerverhaltens sollte nicht nur während der Tests im Usability-Labor vorgenommen werden, sondern auch für die im Web freigeschaltete Anwendung. So kann rechtzeitig eine Veränderung im Benutzerverhalten erkannt und mit Verbesserungen des Webangebotes reagiert werden. Der Usability-Test ist eine permanente QS-Maßnahme im laufenden Betrieb.

Auch wenn Tests in einem Usability-Labor durchgeführt wurden, kann eine Online-Umfrage nach dem Launch interessant sein, um zu prüfen, ob das Usability-Labor noch auf dem aktuellen Stand ist.

Interessante Informationen und weiterführende Links zur Usability sind beim „Informationsdienst Arbeit und Gesundheit“ [URL: Ergo] zu finden.

10.4 Zugänglichkeitstest

Nicht jeder Benutzer kann die PC-Arbeitsmittel wie Tastatur oder Maus einsetzen. Einige können nur schwer oder gar nicht sehen oder hören, einige benutzen einen Sprach-Browser. Deswegen muss sichergestellt werden, dass die Inhalte einer Website für behinderte Benutzer zugänglich sind.

Das W3C (World Wide Web Consortium, [URL: W3C]) setzt die Standards für die Zugänglichkeitsrichtlinien, die insbesondere im öffentlichen Bereich unter dem Begriff „barrierefreies Web“ gefordert werden, fest.

Eine Anwendung ist **barrierefrei**[7], wenn sie für behinderte Menschen in der allgemein üblichen Weise, ohne besondere Erschwernis und grundsätzlich ohne fremde Hilfe zugänglich und nutzbar ist.

► Der **Zugänglichkeitstest** stellt den barrierefreien Zugriff auf die Inhalte eines Webangebotes bzw. die barrierefreie Nutzung einer Applikation sicher.

[7] Barrierefreiheit s. Kap. 2.2

10.4.1 Zugänglichkeitsanforderungen

Die Web Accessibility Initiative (WAI) des W3C [URL: W3C-WAI] beschreibt die Web Content Accessibility Guidelines (WCAG) in Version 2.0. Die deutsche Übersetzung dieser Richtlinien für barrierefrei Webinhalte ist unter [URL: WCAG-de] nachzulesen.

Die Prüfkriterien zum Zugänglichkeitstest sind von der WAI in die **Konformitätsstufen** A, AA und AAA unterteilt; wir sagen im Sinne der Risikoanalyse, dass sie priorisiert sind:

- **Stufe A** (Priorität 1): Für eine Konformität auf Stufe A muss die Webseite alle Erfolgskriterien der Stufe A erfüllen oder es wird eine konforme Alternativversion[8] zur Verfügung gestellt.

  Zum Beispiel muss Bildschirmflackern vermieden werden (nicht mehr als drei Blitze pro Sekunde), um beim Benutzer keine epileptischen Anfälle auszulösen.
- **Stufe AA** (Priorität 2): Für eine Konformität auf Stufe AA muss die Webseite alle Erfolgskriterien der Stufen A und AA erfüllen oder es wird eine Stufe AA-konforme Alternativversion zur Verfügung gestellt.

  Zum Beispiel muss für Benutzer, die eine Website schwarz-weiß betrachten oder eine Rot-Grün-Schwäche haben, der Kontrast von Vorder- zu Hintergrund ausreichend sein, d. h. das Kontrastverhältnis der Schriftfarbe zur Hintergrundfarbe ist mindestens 4,5 zu 1.

  By the way: Zehn von hundert Männern haben eine Rot-Grün-Schwäche und eine von hundert Frauen.
- **Stufe AAA** (Priorität 3): Für eine Konformität auf Stufe AAA muss die Webseite alle Erfolgskriterien der Stufen A, AA und AAA erfüllen oder es wird eine Stufe AAA-konforme Alternativversion zur Verfügung gestellt.

  Zum Beispiel sind *alle* Funktionen ohne Ausnahme nur mit der Tastatur ausführbar.

✓ Checklisten Zugänglichkeit WCAG 2.0 (extern)
Die schweizer „Autorengruppe Accessibility Checkliste 2.0“ [URL: WCAG-ch] stellt die Prüfpunkte der WCAG 2.0 als Checklisten im Word- und Excel-Format auf [URL: WCAG-ch-CL] aktuell zur Verfügung. Daher werden die Inhalte an dieser Stelle nicht aufgeführt.

In Deutschland legt die Verordnung zur Schaffung barrierefreier Informationstechnik nach dem **Behindertengleichstellungsgesetz (BITV) 2.0** die Zugänglichkeitsrichtlinien für Webinhalte fest, s. [URL: BITV]. In der Anlage 1 der BITV werden Anforderungen festgelegt, die sich an den WCAG 2.0 orientieren, aber nur in zwei Prioritätsstufen unterteilt sind. Die BITV 2.0. ist für öffentliche Institutionen rechtsverbindlich.

Eine sehr gute und vollständige Beschreibung der BITV 2.0 sowie weiter Informationen liefert der **BITV-Lotse** des Bundes [URL: BITV-Lotse].

[8] Was jeweils eine „konforme Alternativversion“ ist, wird in [URL: WCAG-de] beschrieben.

Zum Beispiel Bilschirmflackern

Ein Beispiel eines Prüfpunktes der Priorität I der BITV 2.0 ist die Vermeidung von Bildschirmflackern:

- Anforderung 2.3: Inhalte sind so zu gestalten, dass keine epileptischen Anfälle ausgelöst werden.
- Bedingung 2.3.1: Dreimaliges Aufblitzen – Unterschreiten der Schwellenwerte:

Webseiten enthalten keine Elemente, die in einem Zeitraum von einer Sekunde häufiger als dreimal aufblitzen, es sei denn, das Aufblitzen liegt unterhalb der "general flash"- oder "red flash"-Schwelle.

Das negative Prüfergebnis einer Webseite zu diesem Prüfpunkt zeigen das Beispiel in Abb. 10.5 und der folgende Zwischenfall, der 2007 durch die Presse ging (Zitat): [9]

Olympia-Logo löst epileptische Anfälle aus!
„6. Juni 2007 DPA/JR: Seit der Vorstellung des Logos für die Olympischen Sommerspiele in der englischen Hauptstadt reißt die Kritik nicht ab, … Nun musste es sogar von der offiziellen Homepage entfernt werden, weil es offenbar gesundheitsgefährdende Nebenwirkungen hat. Details der Animation hatten 18 epileptische Anfälle ausgelöst, ehe das Logo entfernt wurde … Das Logo basiert auf der Jahreszahl 2012 und enthält die fünf olympischen Ringe sowie den Schriftzug London. Es kann in vier Farben verwendet werden: pink, blau, grün und orange.“ (Internet: www.london2012.org)

10.4.2 Werkzeuge für den Zugänglichkeitstest

Das Bundesverwaltungsamt stellt das Online-Werkzeug **BaNu (Barrieren finden, Nutzbarkeit sichern)** kostenlos zur Verfügung, [URL: BaNu]. Mit BaNu können online Prüfplane erzeugt werden, die festlegen, welche Webseiten nach welchen Kriterien geprüft werden. Die Prüfungen können online abgearbeitet und ausgewertet oder via PDF ausgedruckt werden. Zudem gibt BaNu für jeden Prüfschritt explizite Erläuterungen und Hinweise, wie und mit welchem Tool die Prüfung durchzuführen ist.

Werkzeuge für Zugänglichkeitsprüfungen sind zum Beispiel bei der WAI unter [URL: WAI-Tools] oder auf der Di-Ji Tool-Webseite [URL: DiJi-Tools] aufgeführt. Einige der i. d. R kostenlosen Tools werden im Folgenden vorgestellt.

Einen Online-Dienst zur Überprüfung einer Website bietet **WAVE** das Web Accessibility Evaluation Tool von WebAIM [URL: WAVE]. WAVE stellt die zu prüfenden Webseiten mit erläuternden Symbolen dar, die je nach Schwere des Verstoßes gegen die Zugänglichkeitsrichtlinien farbig markiert sind. Eine Kontrastanalyse wird ebenfalls durchgeführt. WAVE gibt es auch als Firefox-Plugin.

[9] Quelle: http://www.welt.de/sport/article925925/Olympia-Logo-loest-epileptische-Anfaelle-aus.html

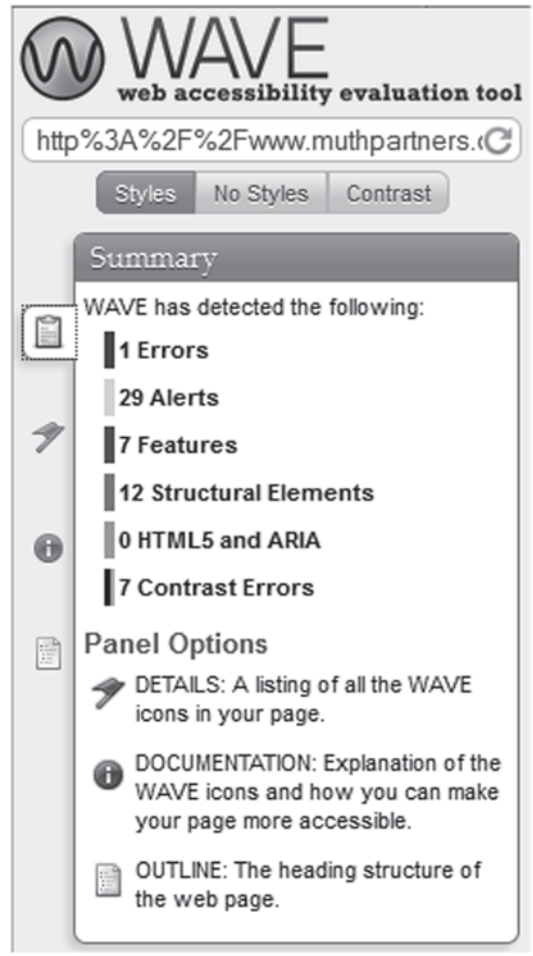


Abb. 10.5 WAVE Prüfprotokoll

Mehrere Tools werden vom Aktionsbündnis für barrierefreie Informationstechnik (AbI) empfohlen, wie die Web Accessibility Toolbar, der Barrierenprüfer A-Prompt und der Farbkontrast-Analyzer, [URL: AbI-Tools].

Mit der im Abschn. 9.8.2 erwähnten **Web Accessibility Toolbar (WAT)** können Zugänglichkeitsprüfungen direkt oder mit aufrufbaren Drittwerkzeugen durchgeführt werden. Allerdings ist dieses Werkzeug nur für den MS Internet Explorer verfügbar.

A-Prompt dient unter Berücksichtigung der Zugänglichkeitsrichtlinien der Überprüfung und Korrektur von HTML-Dokumenten.

Abbildung 10.6 zeigt ein Anwendungsbeispiel mit dem **Farbkontrast-Analyzer**. Darin ist zu sehen, dass die untersuchte Webseite nicht den Anforderungen der Barrierefreiheit genügt, weil die Farbdifferenz der ausgewählten Farben nicht hinreichend ist. Der Farbkontrast-Analyzer kann natürlich auch für beliebige Apps sinnvoll eingesetzt werden.

Zur visuellen Überprüfung der Zugänglichkeit einer Website kann der textbasierte Browser **Lynx** [URL: Lynx] eingesetzt werden. Mit Lynx wird schnell sichtbar, ob eine

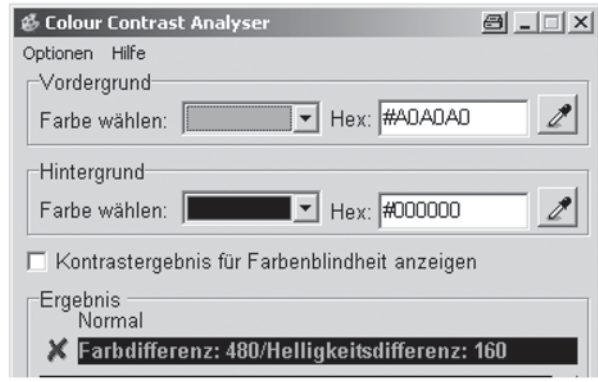


Abb. 10.6 Farbkontrast-Analyzer

Abb. 10.7 Lynx-Browser

komplizierte Navigation oder grafische Spielereien die Website für Behinderte unlesbar machen. In dem Beispiel in Abb. 10.7 wird deutlich, dass in der aufgerufenen Webseite der Umlaut „ä“ (im Wort „nächsten“) nicht richtig angezeigt wird und dass kein alternativer Text für die Grafik angegeben ist (in der eckigen Klammer erscheint nur der Name der gif-Datei), was nach der Checkliste des WAI ein Fehler der Priorität 1 ist.

10.4.3 Qualitätsanforderungen zum Zugänglichkeitstest

Der Zugänglichkeitstest prüft das Qualitätsmerkmal Barrierefreiheit (Accessibility). Die konkreten Qualitätsanforderungen werden aus den Prioritäten der oben vorgestellten Prüfkriterien abgeleitet.

Wird zum Beispiel die Checkliste des WCAG 2.0 zu Grunde gelegt, kann die Qualitätsanforderung an einen Webauftritt die Erfüllung der Konformitätsstufe A zu 100 % und die der Stufe AA zu 80 % sein.

Mit BaNu erstellte Prüfplane stellen ebenfalls gute Anforderungen für einen Zugänglichkeitstest dar; hier wird der Erfüllungsgrad der Prüffragen vorgegeben und gemessen.

10.4.4 Empfehlungen zum Zugänglichkeitstest

BaNu-Prüfpläne als Vertragsbestandteil
Mit BaNu können Prüfplane für den Zugänglichkeitstest einfach und schnell erzeugt werden und als Bestandteil von Verträgen mit externen – und internen – Dienstleistern dienen.

AAA ist nicht immer machbar
Eine Empfehlung seitens des WAI sei erwähnt (Zitat): „Es wird nicht empfohlen, Konformität auf Stufe AAA als allgemeine Richtlinie für komplette Websites zu fordern, da es bei manchen Inhalten nicht möglich ist, alle Erfolgskriterien der Stufe AAA zu erfüllen."

Zugänglichkeit für PDF-Dokumente
Wenn PDF-Dokumente von einer Webseite aufgerufen werden können, sollten auch diese barrierefrei sein. Für PDF-Dokumente gibt es ebenfalls definierte Zugänglichkeitskriterien, zum Beispiel bei [URL: BaNu] in der Rubrik Prüfkataloge – PDF. Bei [URL: WCAG-ch-PDF] ist die Freeware PDF Accessibility Checker (PAC 2) zu finden.

Tooleinsatz bei Entwicklung
Der Zugänglichkeitstest kann von Testern ohne Spezialkenntnisse durchgeführt werden. Es bietet sich an, die Tests von den Entwicklern während der Programmierung mit Werkzeugen durchführen und protokollieren zu lassen. Die Qualitätssicherung überprüft dann vor der Produktfreigabe die Testprotokolle und führt zusätzlich die visuelle Prüfung mit einem geeigneten Browser durch, zum Beispiel mit Lynx.

10.5 Auffindbarkeitstest

Weil das beste Webangebot nichts nutzt, wenn es von niemandem gefunden wird, ist der Auffindbarkeitstest wichtig.

Auffindbarkeit ist die Eigenschaft einer Website, von der Zielgruppe im Internet einfach gefunden werden zu können. Das bedeutet für eine Website, dass die Webadresse (URL = Uniform Ressource Locator) einen sprechenden, leicht zu merkenden Namen hat, und dass Suchmaschinen die Seiten einer Website für bestimmte Stichworte unter den ersten Treffern anzeigen.

► Der **Auffindbarkeitstest** prüft, ob eine Website durch direkte Eingabe einer URL im Browser oder durch Eingabe themenbezogener Stichworte in Suchmaschinen problemlos gefunden wird.

10.5.1 Namensgebung der Webadresse

Die für die Namensgebung einer Webadresse (Domain-Name) zu beachtenden Regeln können in einer kurzen Checkliste „Webadresse“ festgehalten werden:

✓ **Checkliste Webadresse**
- Ist die Webadresse sprechend und möglichst kurz?
- Ist die Webadresse einprägsam?
- Ist (zumindest) das wichtigste Schlüsselwort enthalten?
- Sind Wörter durch Bindestriche getrennt?
- Werden nicht gängige Abkürzungen vermieden?
- Sind alternative Webadressen belegt, die auf die eigentliche URL verweisen, zum Beispiel mit einer gängigen Abkürzung?
- Sind ähnlich klingende URLs belegt, die zum Beispiel bei Schreibfehlern auf die eigentliche URL verweisen?
- Hat die Webadresse einen sinnvollen Bezug zum Unternehmen oder zur angebotenen Dienstleistung?

Diese Fragen lassen sich recht leicht überprüfen, wogegen die im Folgenden beschriebene Optimierung einer Website für Suchmaschinen eine Wissenschaft für sich ist.

10.5.2 Suchmaschinenoptimierung

Die Methoden der **Suchmaschinenoptimierung (SEO** = Search Engine Optimization) sorgen dafür, dass Webseiten für bestimmte Suchbegriffe in der Ergebnisliste von Suchmaschinen möglichst weit oben stehen. Bei der Programmierung einer Website muss darauf geachtet werden, dass sie von den Instrumenten der Suchmaschinen vollständig erfasst und optimal ausgewertet werden kann. Suchmaschinen analysieren die Verlinkungen auf eine Website von außen, die Seiteninhalte und die Link-Texte, die auf die Website verweisen, und bestimmen so die Suchwörter, unter welchen die Website gefunden wird.

Sollen die Webseiten für bestimmte Stichworte von Suchmaschinen mit hoher Priorität versehen und in der Trefferliste möglichst weit oben angezeigt werden, müssen die Inhalte der einzelnen Seiten für diese Stichworte optimiert werden.

Für das **Ranking** der Suchmaschinen, d. h. die Sortierung der Treffer nach Relevanz, spielen sowohl die Inhalte als auch die Aktualität des Webauftritts eine Rolle. Daher muss

das Informationsangebot einer Website nach ihrer Freigabe regelmäßig überarbeitet und erweitert werden.

Die Überprüfung einer Website unter dem Gesichtspunkt der Suchmaschinenoptimierung besteht aus zwei Phasen:

1. Phase: Parallel zur Entwicklung der Webseiten prüfen Experten, ob das Design, der Programmcode und die Inhalte eines Webauftrittes für Suchmaschinen hinreichend optimiert sind. Dazu werden Checklisten und/oder Reviews eingesetzt. Zuvor müssen Checklisten zur Code-Inspektion um die Anforderungen zur Suchmaschinenoptimierung ergänzt werden.
2. Phase: Im späteren Betrieb wird das Ranking der Webseiten regelmäßig überprüft. Das ist notwendig, weil das Ranking von den Suchmaschinen permanent neu berechnet wird und ggf. Maßnahmen ergriffen werden müssen, wenn sich das Ranking einer Webseite verschlechtert.

Die einzuhaltenden Standards und Empfehlungen zur Suchmaschinenoptimierung sind unter anderem auf den entsprechenden Seiten der Suchmaschinen wie zum Beispiel von Google zu finden [URL: SEO-Google]. Sie sollten in die entsprechenden Checklisten zur Inspektion einer Website einfließen. Hier eine kleine Auswahl von wichtigen Prüfpunkten:

✓ Checkliste Barrierefreiheit und SEO-Freundlichkeit

CSS

- Sind die Seiten nicht zu breit, d. h. werden bei geringen Auflösungen horizontale Scroll-Balken vermieden?
- Sind Breiten in „em“ (relative Maßeinheit) und nicht in „px“ (Pixel, absolute Maßeinheit) angegeben?[10]
- Werden wichtige Textpassagen mit dem -Tag hervorgehoben?
- Werden Tabellen über DIV-Layouts definiert?

Links und Navigation

- Sind die Link-Texte aussagekräftig und enthalten die gewünschten Suchwörter/Schlüsselwörter?
- Sind alle Links als Textlinks angelegt, d. h. sind Links auf Grafiken und Bildern vermieden?
- Bestehen Link-Texte nicht aus ganzen Sätzen?
- Werden keine Symbole in den Link-Texten verwendet (wie z. B. „<“, „>“, „&“)?
- Ist die Navigation nicht mit JavaScript und nicht mit Flash realisiert?
- Ist die Navigation – zumindest alternativ – über CSS-formatierte Text-Links realisiert?
- Werden keine Framesets verwendet?
- Sind die wichtigen Schlüsselwörter im Pfadnamen enthalten?

[10] Barrierefreie Seiten sind frei einstellbar.

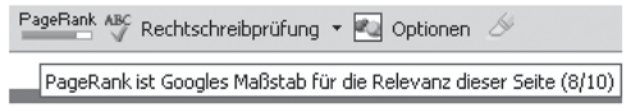


Abb. 10.8 PageRank

Bilder und Grafiken
- Besitzt jedes Bild und jede Grafik ein <alt>-Tag mit einem Stichwort zum Bild?
- Besitzt jedes <alt>-Tag mindestens ein Schlüsselwort?

Seitentitel
- Sind die Seitentitel aussagekräftig?
- Sind die relevanten Schlüsselwörter im Seitentitel enthalten?
- Ist der Seitentitel maximal 65 Zeichen lang?

Inhalt
- Ist die Seitenbeschreibung maximal 65 Zeichen lang?
- Ist die Seite auf zwei Schlüsselwörter ausgerichtet?
- Sind die relevanten Schlüsselwörter in der Seitenbeschreibung enthalten?
- Sind die relevanten Schlüsselwörter drei bis vier Mal je 100 Wörter Text enthalten?
- Ist der Text einer Seite mindestens 300 Zeichen groß?
- Sind die Schlüsselwörter im Namen verlinkter Dateien enthalten?
- Ist der Text durch Überschriften und Unterüberschriften strukturiert (<H1>-, <H2>-, <H3>-,…, -Tags)?

10.5.3 Werkzeuge für den Auffindbarkeitstest

Die Position eines Suchergebnisses in einer Trefferliste lässt sich direkt durch die Eingabe des Stichwortes in einer Suchmaschine überprüfen.

PageRank von Google zeigt in der Google Toolbar die Beliebtheit einer Webseite bei Google an (s. Abb. 10.8). Die Grundlage für das Maß bilden die von anderen Webseiten hereinführenden Links auf diese Webseite (Inbound Links).

Es gibt eine Vielzahl von i. d. R kostenpflichtigen **SEO-Tools**, die im Web unter den entsprechenden Stichworten wie „Page Ranking Tool“, „Ranking Report“ oder „SEO-Tool“ mit Suchmaschinen gefunden werden können.

Die Prüfung einer Webseite auf Einhaltung von SEO-Kriterien (s. Abb. 10.9), kann zum Beispiel mit dem Werkzeug **SenSEO** [URL: SenSEO] durchgeführt werden das als Firefox Add-on läuft. SenSEO liefert zudem Metriken zu Objekten und Schlüsselwörtern einer Webseite (s. Abb. 10.10).

Abb. 10.9 SEO-Prüfung mit SenSEO

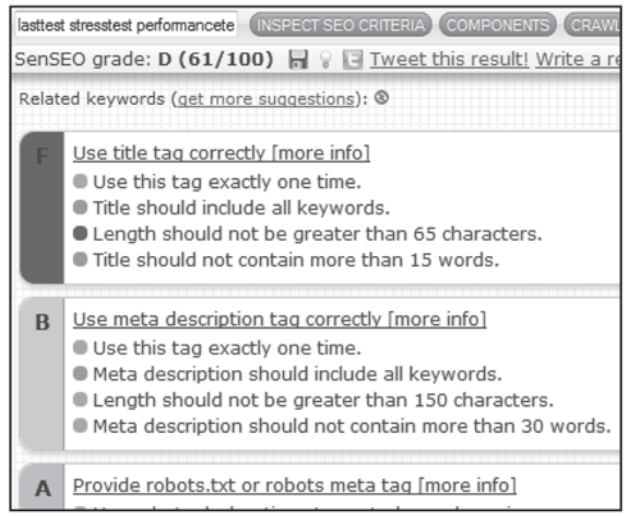


Abb. 10.10 SEO-Metriken

number of elements	395
number of words	481
element-word ratio	1:82
keyword matches	3
keyword density	0.62

Ein kostenloser **Backlink Checker** wird unter [URL: BacklinkTest] angeboten. Mit diesem Werkzeug werden die Webseiten aufgelistet, die einen Link auf die zu prüfende Website gesetzt haben. Eine Liste mit Beschreibungen von Backlink Checkern stellt Karl Kratz unter [URL: BacklinkChecker] zur Verfügung.

10.5.4 Qualitätsanforderungen zum Auffindbarkeitstest

Die Einhaltung der Richtlinien zur Namensgebung von Webadressen kann einfach und ohne Werkzeugunterstützung anhand der vorgegebenen Checkliste „Webadresse“ nachgewiesen werden.

Das Ranking der Webseiten und die gewünschte Position innerhalb der Trefferliste zu bestimmten Suchwörtern kann in Zahlen gemessen werden. Dazu müssen die Qualitätsanforderungen konkret vorgegeben sein, wie in den folgenden Beispielen.

Qualitätsanforderungen zum Ranking
Das Ranking einer Webseite muss nach ihrer Freischaltung bei der Suchmaschine „Xyz“ folgende Werte auf der 10-wertigen Ranking-Skala erreichen:

- ab der Woche 6 den Ranking-Wert 3
- ab der Woche 10 den Wert 5
- ab der Woche 14 und folgenden mindestens den Wert 6

Qualitätsanforderungen zur Suchmaschinenpositionierung
Eine ehrgeizige Qualitätsanforderung zur Positionierung einer Website in den Trefferlisten von Suchmaschinen wäre folgende:

Zum Stichwort „Webtesting" rangiert die Webseite zu diesem Buch im Webauftritt des Verlages vier Wochen nach ihrer Freischaltung und die 24 darauf folgenden Monate bei den beiden Suchmaschinen Xxx und Yyy unter den Top 10.

10.5.5 Empfehlungen zum Auffindbarkeitstest

Der Auffindbarkeitstest ist als permanente Aktivität über die gesamte Lebensdauer einer Website durchzuführen, sofern der Auftraggeber die Qualitätsanforderungen zur Auffindbarkeit konkret definiert hat. Weil zur Suchmaschinenoptimierung besonderes Fachwissen über Webdesign, Marketing und die Algorithmen der Suchmaschinen benötigt wird, ist es ratsam, diese Dienstleistung – zumindest in der Startphase – von Experten einzukaufen.

Mit der Erfüllung der Kriterien zur Zugänglichkeit wird auch die Auffindbarkeit verbessert, weil Suchmaschinen eine Website so „sehen", wie sie sich im Lynx-Browser darstellt. Barrierefreiheit eines Webauftritts wirkt sich positiv auf seine Auffindbarkeit aus.

Um eine Website bekannt zu machen, sollte sie natürlich bei den wichtigsten Suchmaschinen angemeldet werden. Die Anmeldung kann direkt manuell bei den Anbietern der Suchmaschinen oder automatisiert durch Programme, die von SEO-Dienstleistern angeboten werden, erfolgen. Eine regelmäßige Wiederanmeldung verbessert die Auffindbarkeit und das Ranking der Webseiten.

Je mehr Backlinks zur eigenen Website führen, desto besser wird sie aufgefunden. Lassen Sie dafür ihre Website bei möglichst vielen Freunden, Partnern, Kunden und Foren verlinken.

Aber vermeiden Sie Link-Farmen, diese können sich bei Suchmaschinen kontraproduktiv auswirken.

► Eine gute Website ist barrierefrei und suchmaschinenfreundlich!

10.6 Zusammenfassung

- Ziel des Benutzbarkeitstest ist die Sicherstellung der Qualitätsmerkmale Bedienbarkeit, Erlernbarkeit, Verständlichkeit sowie der webspezifischen Qualitätsmerkmale Auffindbarkeit, Barrierefreiheit und Rechtskonformität einer Webanwendung. Ziel ist nicht die Überprüfung der fachlich angeforderten Funktionalitäten, die in den Tests zur Funktionalität stattfindet.

- Die Benutzbarkeit einer Webapplikation wird durch Content-, Oberflächen-, Browser-, Usability-, Zugänglichkeits- und Auffindbarkeitstest nachgewiesen.
- Der Content-Test ist eine Erweiterung des Dokumententests für Webauftritte. Er stellt für eine Website sicher, dass die Benutzererwartungen erfüllt werden, der Aufklärungspflicht genüge getan ist und keine rechtlichen Verstöße vorliegen.
- Der Content-Test wird von Rechts- und Marketingexperten anhand von Checklisten in Reviews oder schriftlichen Stellungnahmen an einem Prototypen durchgeführt.
- Der Oberflächentest überprüft die Darstellung, die Erreichbarkeit und allgemeingültige Funktionalitäten der Anwendung auf der Browser-Oberfläche.
- Anwendungsspezifische Funktionen, für die mit den entsprechenden Methoden fachliche Testfälle erstellt werden, bleiben beim Oberflächentest unberücksichtigt.
- Eine Checkliste liefert die Testfälle zum Oberflächentest von Webanwendungen und standardisiert die Tests.
- Im Usability-Test wird der Benutzer im Usability-Labor beobachtet. Alternativ oder ergänzend dazu liefern mündliche und schriftliche Befragungen Informationen zur Gebrauchstauglichkeit einer Webanwendung. Der Usability-Test wird von Fachleuten durchgeführt.
- Der Usability-Test kann für Websites, Web-Apps und Mobile-Apps durchgeführt werden.
- User Tracking Tools und Eye Tracker liefern Informationen über das Benutzerverhalten und den Datenverkehr auf einer Website. Die Gebrauchstauglichkeit einer Website muss während des produktiven Betriebes überwacht werden.
- Der Zugänglichkeitstest stellt den barrierefreien Zugriff auf eine Anwendung sicher. Die Anforderungen zur Barrierefreiheit sind international durch die Web Accessibility Initiative (WAI) und national durch die Barrierefreie Informationstechnik-Verordnung (BITV) festgelegt und priorisiert.
- Auffindbarkeit wird durch richtige Namensgebung der URL und Optimierung der Webseiten für Suchmaschinen durch Experten erreicht. Eine barrierefreie Webseite ist für Suchmaschinen gut auffindbar.

Tests zur Änderbarkeit und Übertragbarkeit 11

Das Problem zu erkennen ist wichtiger, als die Lösung zu erkennen, denn die genaue Darstellung des Problems führt zur Lösung.

Albert Einstein (1879–1955)

Eine Software, die nach vorgegebenen Programmierstandards und modular entwickelt ist, kann mit geringerem Aufwand geändert, erweitert und auf andere Systeme übertragen werden, als unstrukturiert entwickelte. Die Prüfung des Qualitätsmerkmals Änderbarkeit wird in Code-Analysen vorgenommen. Die Eigenschaft der Übertragbarkeit einer Anwendung auf unterschiedliche Zielsysteme wird im Installationstest getestet.

Gute Änderbarkeit und Übertragbarkeit einer Software beruht auf einem entsprechenden Systemdesign und strukturiertem Programmcode. Daher werden diese beiden entwicklungsnahen Qualitätsmerkmale in diesem Kapitel zusammen behandelt.

11.1 Code-Analysen

Code-Analysen sind als statische Prüfungen spezielle Ausprägungen des Dokumententests (s. Kap. 8.2), die auf Code-Ebene eingesetzt werden. Die zu prüfenden Dokumente enthalten entweder Pseudocode, welcher der Programmgenerierung dient, oder den programmierten Sourcecode. Zu den Code-Analysen gehören der Code-Walkthrough und die Code-Inspektion, die beide in Review-Sitzungen durchgeführt werden, der Schreibtischtest, der eine Ausprägung der schriftlichen Stellungnahme ist, und die statische Code-Analyse, die mit Werkzeugen vorgenommen wird.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_11

11.1.1 Code-Walkthrough

► Ein **Code-Walkthrough** ist eine visuelle Überprüfung von Pseudo- oder Sourcecode. Dabei werden Programmabläufe anhand von Beispieldaten nachvollzogen.

Bei einem Code-Walkthrough erklärt der Autor die Ablauflogik des Moduls. Sehr effektiv ist das Verfahren, wenn das Programm nicht (vor −)gelesen wird, sondern das Testteam anhand ausgewählter Testfälle Computer spielt. Die Verarbeitung der Daten wird im Programmcode „manuell" nachvollzogen. Durch Diskussionen der Gruppe mit dem Programmierer werden Fehler und Unvollständigkeiten aufgedeckt. Während der Sitzung werden keine Lösungen diskutiert, sondern Fehlerzustände „nur" aufgedeckt und dokumentiert.

Die Testszenarien und Beispieldaten für einen Code-Walkthrough werden vor der Sitzung mit Blackbox-Verfahren (s. Kap. 4.1) entworfen.

11.1.2 Code-Inspektion

► Eine **Code-Inspektion** ist eine visuelle Überprüfung von Sourcecode. Dabei wird das Programm mit Hilfe von Checklisten zu Programmierstandards und häufig vorkommenden Fehlern analysiert.

In der Inspektionssitzung stellt das Team dem Autor Fragen, wobei durch die auftretenden Diskussionen Fehler und Unvollständigkeiten aufgedeckt werden. Zusätzlich wird das Programm mit Hilfe einer Checkliste geprüft.

Die folgende Checkliste zur Code-Inspektion ist, obwohl objektorientiert ausgerichtet, programmiersprachenneutral. Sie ist als „Ideenspender" zu sehen, um daraus eigene, programmiersprachenspezifische Checklisten zu entwickeln.

✓ Checkliste Code-Inspektion
Konformität
- Entsprechen alle Namen den gültigen Konventionen?
- Sind Namensgebungen eindeutig und selbstbeschreibend für:
  - Dateien?
  - Grafiken?
  - Links?
  - Menüs?
- Sind die Programmierrichtlinien eingehalten?
- Sind die Layout-Konventionen eingehalten?
- Sind wiederverwendbare Module ausgelagert?

Namenskonventionen

- Haben Variablen-, Konstanten-, Methoden- oder Klassennamen ausgeschriebene Namen, die den Inhalt oder die Aufgabe beschreiben?
- Besitzen die Namen der Methoden, die Klassenvariablen mit neuen Werten belegen, das Präfix „set“?
- Besitzen die Namen der Methoden, mit denen der Wert von Klassenvariablen ausgelesen wird, das Präfix „get“?
- Gibt es keine Namen, die sich nur in der Groß-/Klein-Schreibweise voneinander unterscheiden?
- Beginnen Variablen- und Methodennamen immer mit einem Kleinbuchstaben?
- Beginnen keine Variablennamen mit einem „_“ oder „$“.
- Werden Variablennamen für unterschiedliche Aufgaben oder Datentypen innerhalb einer Methode nicht mehrfach verwendet?
- Entsprechen die Namen der Variablen für mathematische Aufgaben den in der Mathematik geläufigen Namen?
- Beginnen Klassennamen mit einem Großbuchstaben?
- Bestehen Konstanten nur aus Großbuchstaben, Ziffern und dem Unterstrich?

Kommentare

- Ist der Code ausreichend kommentiert?
- Beschreiben die Code-Kommentare die inhaltliche Wirkung der Anweisungen und nicht nur die reine Syntax?
- Ist jeder Autor, der an einer Klasse/ einem Modul Veränderungen vorgenommen hat, mit dem vollständigen Namen, dem verwendeten Kürzel und ggf. der Firma angegeben?
- Ist „ineffizienter Code“, der bewusst codiert wurde, mit Autor, Datum und Begründung versehen?
- Geht die Methodenbeschreibung auf alle bekannten Randbedingungen und Sonderfälle ein, sofern diese existieren?
- Sind Programmmodifikationen in einer Änderungshistorie mit Datum, Bearbeiter und Beschreibung festgehalten?

Berechnungen und Vergleiche

- Werden komplexe Ausdrücke durch zusätzliche Klammerpaare übersichtlich gestaltet?
- Wird eine Konstante stets für denselben Zweck verwendet?
- Werden Berechnungen nur mit numerischen Feldern ausgeführt?
- Sind bei Berechnungen mit unterschiedlichen Datentypen die Konvertierungen richtig?
- Haben bei Vergleichen alle Operanden den gleichen Datentyp bzw. sind die Konvertierungsregeln korrekt?

- Werden Datumsfelder richtig konvertiert?
- Wird Division durch 0 abgefangen?
- Kann es keine Über- oder Unterläufe geben?
- Kann es keine Rundungsfehler geben?
- Sind alle Ausdrücke und Formeln richtig?

Datenreferenzen
- Werden alle Felder explizit definiert?
- Werden alle Variablen richtig initialisiert?
- Werden alle Tabellen richtig initialisiert?
- Liegen alle Indexwerte innerhalb der Tabellengrenzen?
- Sind Indizes ganzzahlig definiert?
- Sind Zuweisungen bzgl. der Zielvariablen sinnvoll?

Steuerung
- Kommt jede Schleife zu einem Ende?
- Werden Schleifenzähler richtig initialisiert?
- Können Schleifenzähler nicht zu groß werden?
- Gibt es keine „toten“ Bedingungszweige?

Schnittstellen
- Sind Anzahl und Reihenfolge der Übergabeparameter korrekt?
- Stimmen Parameter und Argumente in Typ und Länge überein?
- Sind die gleichen Parameter immer in der gleichen Reihenfolge definiert, wenn sie von mehreren Methoden verwendet werden?
- Sind globale Variablen notwendig und über alle Module konsistent?
- Werden Eingabeparameter nicht verändert?
- Stimmen Ein/Ausgabe-Bereiche mit den Satzlängen überein?
- Werden Cursor- und Dateiendebedingungen erkannt und richtig behandelt?
- Werden Cursor und Dateien rechtzeitig und korrekt eröffnet?
- Werden Cursor und Dateien geschlossen?

Fehlerbehandlung
- Werden alle Eingabedaten auf Gültigkeit geprüft, falsche Eingaben erkannt und richtig behandelt?
- Werden alle Datenbankfehler richtig erkannt und behandelt?
- Werden alle Systemfehler richtig erkannt und behandelt?
- Werden Commit-Points richtig gesetzt?
- Sind Fehler- und Hinweistexte korrekt und sinnvoll?

Website
- Hat jedes Unterverzeichnis der Website eine eigene Index-Seite (Index.htm)?
- Gibt es eine erklärende HTML-Seite, die bei Fehlern angezeigt wird?
- Sind Grafiken im Code mit Größenangaben versehen?
- Sind Menüs mit Default-Werten belegt?
- Wird der Zielinhalt eines internen Links in dem Fenster geöffnet, in dem sich der interne Link befindet?
- Wird der Zielinhalt eines externen Links in einem neuen Fenster geöffnet?
- Sind die Webseiten für Suchmaschinen optimiert (s. Abschn. 10.5.2)?

11.1.3 Schreibtischtest

Code-Walkthroughs und Code-Inspektionen sind aufwendige Review-Arten. Eine Alternative ist der Schreibtischtest.

► Der **Schreibtischtest** ist eine Ein-Personen-Code-Inspektion kombiniert mit einem Ein-Personen-Walkthrough.

Beim Schreibtischtest wird Sourcecode an Hand von Beispielen und Checklisten durch eine Person überprüft, die nicht der Autor des Dokumentes ist. Fragen, Fehler und Verbesserungsvorschläge werden notiert und anschließend mit dem Autor besprochen. Der Schreibtischtest fasst die Methoden Code-Walkthrough und Code-Inspektion in einer „Light"-Ausprägung zusammen, was diese Form der Qualitätssicherung effektiv und relativ kostengünstig macht.

11.1.4 Statische Code-Analyse durch Werkzeuge

Die Überprüfung von Programmcode auf Einhaltung von Programmierstandards und Richtlinien kann, wie oben beschrieben, mit Hilfe von Checklisten manuell vorgenommen werden. Effizienter wird eine **statische Code-Analyse** von **Code Analyzern** durchgeführt, die folgende Aufgaben erledigen:

- Prüfung der Syntax
- Prüfung auf korrekte und typgerechte Verwendung, Initialisierung und Referenzierung der Variablen und Klassen
- Aufspüren von nicht erreichbarem Code
- Erkennen von Speicherlecks (s. Kap. 12.4)
- Erstellung der Verwendungsnachweise von Variablen und Funktionen (Cross Reference)
- Ermittlung der Komplexität eines Programms

Abb. 11.1 Unicorn

Jeder **Compiler** bietet Möglichkeiten zur statischen Analyse des Quellcodes eines Programms.

Darüber hinaus gibt es für jede gängige Programmiersprache **Code Analyzer**, die Sourcecode nach vorgebbaren Regeln überprüfen, Datenfluss- und Kontrollflussanalysen durchführen und Aussagen über die Komplexität eines Programms machen können. Als Beispiel für Java, JavaScript, XML, XSL sei das Werkzeug **PMD** genannt, [URL: PMD]. Speziell für Java sind Open Source Tools unter [URL: TestToolsJava] in der Rubrik „Code Analyzers" zu finden.

Für Web-Programmiersprachen gibt es kostenlose Hilfsmittel zur Code-Überprüfung. Zum Beispiel bietet das World Wide Web Consortium unter [URL: W3C-validatorMarkup] und [URL: W3C-validatorCSS] einzelne **Validatoren** für HTML, XHTML[GL], CSS[GL] und XML[GL] an, sowie unter [URL: W3C-Unicorn] den Einheitsvalidator **Unicorn** (s. Abb. 11.1).

Unicorn enthält unter anderem eine Prüfungsoption auf Mobiltauglichkeit, die auch der im Abschn. 9.12.1 beschriebene W3C **mobileOK Checker** [URL: W3C-validator MobileOK] überprüft.

Das Programm **HTML Tidy** [URL: Tidy] von Dave Raggett repariert und verbessert sogar HTML-Code. Die Prüftiefe, wie zum Beispiel bei der Analyse der Zugänglichkeitsanforderungen (s. Abschn. 10.4.1), kann in HTML Tidy voreingestellt werden. Erweiterungen dieses Open Source Programms für Windows und andere Betriebssysteme sind neben dem Original im sourceforge.net zu finden. Es gibt allerdings die Einschränkung, dass dynamische HTML-Seiten, wie sie i. d. R. bei umfangreicheren Webanwendungen generiert werden, mit Tidy nicht geprüft werden können.

11.1.5 Qualitätsanforderungen zu Code-Analysen

Die Qualitätsanforderungen zur **Änderbarkeit** und **Übertragbarkeit** werden durch Prüfprotokolle und Checklisten nachgewiesen, die während der Code-Analysen erstellt bzw. ausgefüllt werden.

Eine wichtige Rolle bei der Festlegung der Qualitätsanforderungen spielen die geplante Vermarktung und der Einsatzbereich einer Anwendung. Das bedeutet, dass die konkreten Qualitätsanforderungen von den Antworten auf die folgenden Fragen abhängen:

- Wird die Anwendung nur im eigenen Hause betrieben?
- Soll sie für mehrere Mandanten eingesetzt werden?
- Wird sie an Kunden geliefert, die individuelle Anpassungen beauftragen werden?
- Sind regelmäßige Updates und funktionale Erweiterungen geplant?
- Werden einzuhaltende Standards vom Auftraggeber vorgegeben?
- Wie viele und welche Module müssen wiederverwendbar sein, weil sie in anderen Anwendungen implementiert werden sollen?

Qualitätsanforderungen unter dem Aspekt der **Wiederverwendbarkeit** können lauten:

- Software-Komponenten, die als wiederverwendbar eingestuft sind, müssen einer Code-Inspektion mit mindestens drei Experten unterliegen. Für sie muss die Checkliste „Code-Inspektion“ zu 100 % erfüllt sein und eine automatische Code-Analyse durchgeführt werden, deren Ergebnisbericht keine Fehler und keine Warnungen ausweisen darf.
- Software-Komponenten, die als nicht wiederverwendbar eingestuft sind, müssen einen Schreibtischtest bestehen. Für sie muss vorab eine automatische Code-Analyse durchgeführt werden, deren Ergebnisbericht nur Warnungen und keine Fehler ausweisen darf.

Anhand von **Komplexitätsmaßen** kann die Änderbarkeit eines Programmes eingeschätzt werden: je komplexer, desto schlechter lässt sich der Sourcecode ändern. Komplexitätsmaße, die mit den oben beschriebenen Analyzern bestimmt werden, sind zum Beispiel:

- Die **Schachtelungstiefe** von Bedingungen und Schleifen sollte nicht tiefer als 6 sein.
- Die **zyklomatische Zahl**[1] – sie gibt die Anzahl der linear unabhängigen Pfade in einem Modul an – sollte unter 10 liegen.

[1] Wie die zyklomatische Zahl berechnet wird, ist z. B. in [Spillner_2012] nachzulesen.

11.1.6 Empfehlungen zu Code-Analysen

Statische Code-Analyse zuerst durchführen
Jeder visuellen Code-Analyse sollte eine statische Code-Analyse mit einem Werkzeug vorangehen. Je nach Umfang der statischen Analyse kann in Abhängigkeit der geforderten Qualitätsanforderungen auf eine manuelle Code-Analyse verzichtet werden.

Code-Analysen durch Entwickler
Code-Analysen sind sehr entwicklungsnah und sollten daher im Unit-Test vom Entwickler durchgeführt werden. Die erzeugten Prüfprotokolle werden dann als Testnachweis mit der zu testenden Komponente an das Testteam gegeben.

Checklisten als konstruktive Maßnahme
Die Checklisten zur Code-Inspektion sollten, sofern es diese im Unternehmen noch nicht gibt, von Experten für die eingesetzten Programmiersprachen erarbeitet werden. Als konstruktive Qualitätsmaßnahme werden die Checklisten zum Beginn der Realisierungsphase den Entwicklern zur Verfügung gestellt.

11.2 Installationstest

Web-Apps werden per Definition nicht installiert, aber Browser und Plugins auf dem Client, Software-Komponenten auf dem Server und insbesondere Mobile-Apps auf dem Smartphone. Daher wird der Installationstest zuerst im Allgemeinen beschrieben und dann im Besonderen für Mobile-Apps.

11.2.1 Installationsphasen

Installierbarkeit ist die Fähigkeit eines Software-Produkts in einer festgelegten Umgebung installierbar zu sein. Sie wird am Aufwand gemessen, der zur Installierung der Software in einer festgelegten Umgebung notwendig ist.

Für die **Deinstallierbarkeit** einer Software gelten die entsprechenden Aussagen.

► Der **Installationstest** prüft die Installierbarkeit und Deinstallierbarkeit einer Anwendung.

Der Installationstest stellt sicher, dass Installationsprozesse effizient und fehlerfrei durchgeführt werden können. Zu den Installationsprozessen gehören Erstinstallation, Installation neuer Versionen und Deinstallation der Software.

Für die Installation werden eigenständige Programme, die Installationsroutinen, benötigt. Diese müssen, wie die zu installierende Software selbst, getestet werden. Die Prüfung der Lizenzvergabe gehört ebenfalls zum Installationstest.

Eine Installationsroutine verlangt in der Regel vom „Installateur“ Eingaben und Parametereinstellungen; für diese Eingabemöglichkeiten müssen Testfälle entworfen werden.

Auch technische Belange müssen berücksichtigt werden. Zum Beispiel muss die Klassen-ID eines OLE-Objektes (COM/DCOM) in der Registrierungsdatenbank des Betriebssystems eingetragen sein, damit die Anwendung funktioniert. Oder wenn die Anwendung JavaScript benutzt, muss diese Einstellung im Browser bei der Installation aktiviert werden (s. Kap. 9.8). Eventuell benötigt der Benutzer Administratorrechte zur Installation von Komponenten auf dem Client.

Nicht nur die Erstinstallation, sondern auch die Installation eines Updates ist vor Auslieferung einer neuen Software-Version anhand von Testfällen zu testen.

Ebenso müssen Deinstallationsroutinen, die ebenfalls zum Lieferumfang einer Software gehören, getestet werden. Sie dürfen zum Beispiel keine falschen Daten löschen. Die Deinstallation einer Anwendung kann erforderlich sein, bevor eine neue Version installiert oder wenn die bestehende Version endgültig vom Rechner genommen werden soll.

Die Checkliste „Installationstest“ fasst die zu prüfenden Punkte zusammen:

✓ Checkliste Installationstest

Installation

- Wird der Benutzer ggf. darauf hingewiesen, dass er für die Installation Administratorrechte besitzen muss?
- Prüft die Installationsroutine, ob Hard- und Software des Zielsystems kompatibel sind?
- Werden im Fall der Inkompatibilität Fehlerhinweise erzeugt und wird die Installation korrekt abgebrochen?
- Wird geprüft, ob der von der Anwendung benötigte freie Hauptspeicherplatz vorhanden ist?
- Wird geprüft, ob genügend Speicherplatz auf der Festplatte vorhanden ist?
- Wird bei Abbruch des Installationsvorgangs der ursprüngliche Zustand wieder hergestellt?
- Wird der Benutzer darauf hingewiesen, wenn existierende Dateien mit neuen Versionen überschrieben werden?
- Sind alle Eingaben, die zur Installation notwendig und möglich sind, in Testfällen abgebildet?
- Werden die benötigten Komponenten (ActiveX-Controls, DLLs, Plugins, …) richtig installiert?
- Wird auf bereits installierte Komponenten (ActiveX-Controls, DLLs, Plugins, …), die mit den zu installierenden Komponenten kollidieren können, hingewiesen?
- Werden Browser richtig konfiguriert? (Werden zum Beispiel JavaScripts benutzt, so muss diese Option bei der Installation automatisch aktiviert bzw. angefordert werden.)
- Ist die Installation von allen erforderlichen Medien möglich (CD, DVD, Download vom Webserver, USB-Medium) und wurde sie damit getestet?

Lizenzen
- Werden die Lizenzvereinbarungen dargestellt und müssen sie bestätigt werden?
- Wird die Installation abgebrochen, wenn die Lizenzvereinbarungen nicht bestätigt werden?
- Wird die Lizenzschlüssel-Vergabe geprüft?

Update
- Wird beim Update überprüft, ob die installierte Version kompatibel zur neuen Version ist?
- Werden Benutzer online über neue, bereitstehende Versionen (Updates) informiert?
- Ist das Verfahren zur Verteilung von Updates beschrieben und getestet?
- Funktioniert die Anwendung auch, wenn eine Update-Version übersprungen wurde?
- Werden vor dem Update die vorgefundenen Zustände gesichert?
- Der Benutzer wird beim Update nicht automatisch ohne Hinweismeldung ausgeloggt?
- Werden im Bedarfsfall Sicherungen wieder richtig zurückgespielt?
- Gehen beim Update keine Nutzerdaten verloren?
- Werden Benutzereinstellungen übernommen?
- Werden Datenbank- und Dateiinhalte vollständig und korrekt übernommen?

Deinstallation
- Werden nur zur Anwendung gehörende Dateien gelöscht?
- Werden alle nicht benötigten Dateien, Icons, Menüeinträge und Pfade vollständig entfernt?
- Werden die Registry Keys entfernt?

Für jeden in der Checkliste aufgeführten Themenbereich müssen Testfälle beschrieben werden. Dieses geschieht mit den Methoden der systematischen Testfallerstellung (s. Kap. 4), wobei die Anforderungen nicht nur fachlicher, sondern auch technischer Natur sind.

Die Prozesse, die bei einer Installation durchzuführen sind, werden für den „Installateur“ in einem Installationshandbuch beschrieben. Diese spezielle Benutzerdokumentation muss vor Auslieferung einem Dokumententest (s. Kap. 8) unterzogen werden.

11.2.2 Mobile-Installationstest

Die Checkliste Installationstest gilt auch für Mobile-Apps, die installiert werden müssen (Hybrid- und Native-Apps). Allerdings müssen im mobilen Umfeld spezifische Aspekte berücksichtigt werden. Was passiert zum Beispiel, wenn während der Installation

dem Endgerät der „Saft“ ausgeht, die Netzverbindung plötzlich weg ist oder ein Anruf ‚reinkommt‘?

Folgende Punkte der Checkliste „Mobile-Installationstest“ müssen für installierbare Mobile-Apps getestet werden:

✓ Checkliste Mobile-Installationstest

Allgemeines

- Installiere die Mobile-App bei unterschiedlichen Verbindungsarten (EDGE, GRPS, HSPA, LTE, UMTS, WLAN, Satellit, …).
- Installiere die App bei zu wenig Gerätespeicher.
- Installiere die App auf SD-Karte (sofern vorhanden).

Batteriestand

- Installiere die Mobile-App bei sehr niedrigem Akkustand.
- Schalte während der Installation das Gerät aus und danach wieder ein.
- Sorge dafür, dass während er Installation der Zeitpunkt für die automatische Abschaltung des Gerätes kommt, weil die Batterie leer ist.

Störungen

- Unterbreche den Installationsprozess durch eingehende Störungen (E-Mail, Anruf, …) und kehre zur Installation zurück.
- Installiere die App vom Server bei schlechter Funknetzverbindung.
- Unterbreche die Funknetzverbindung während der Installation vom Server für eine gewisse Dauer und stelle sie dann wieder her. (Gehe zum Beispiel in den Flugmodus.)
- Installiere die App vom Server bei schlechter WLAN-Verbindung.
- Unterbreche die WLAN-Verbindung während der Installation vom Server für eine gewisse Dauer und stelle sie dann wieder her.
- Das Gerät ist via WLAN *und* Mobilfunknetz mit dem Internet verbunden. Nimm während der Installation vom Server eine Verbindungsart weg.
- Entferne während der Installation auf SD-Karte die SD-Karte aus dem Gerät.

Für alle aufgeführten Prüfpunkte müssen Testfälle beschrieben werden, aus denen hervorgeht, wie die technische Situation, bzw. der Störfall herzuleiten ist und wie das System konkret darauf reagieren soll. Der Benutzer muss über die jeweilige Situation informiert werden, z. B. dass und warum die Installation so lange dauert oder warum sie abgebrochen wurde und was die nächsten Handlungsmöglichkeiten sind.

11.2.3 Werkzeuge für den Installationstest

Der Einsatz von **virtuellen Maschinen** (s. auch Abschn. 9.11.4) ermöglicht, dass der Zustand eines Systems wie vor der Installation einfach wiederhergestellt werden kann. Dieser Originalzustand ist mit einer Deinstallation nicht sichergestellt.

Im Installationstest können viele Testfälle mit einem **Capture Replay Tool** (s. Kap. 14.3) ausgeführt werden. Wenn der Installationstest für unterschiedlich konfigurierte Zielsysteme mehrfach durchgeführt werden muss, amortisiert sich der Automatisierungsaufwand sehr schnell.

Testfälle, die Störungen des Prozesses enthalten, werden i. d. R. nur manuell ausführbar sein.

11.2.4 Qualitätsanforderungen zum Installationstest

Die Installierbarkeit ist ein Teilmerkmal des Qualitätsmerkmals Übertragbarkeit. Sie wird durch den Testnachweis aller Testfälle zum Installationstest sichergestellt. Eine Testfallüberdeckung von 100 % ist hier gefordert.

Wenn der Auftraggeber bzw. der Benutzer die Software selbständig installieren muss, müssen Qualitätsanforderungen an die Benutzerfreundlichkeit der Installationsanweisungen und den mit der Installation verbundenen Aufwand gestellt werden.

Dazu könnte von einem Usability-Test zum Installationsprozess gefordert werden, dass 20 Testnutzer mit unterschiedlichen IT-Kenntnissen die Software anhand der Installationsanweisungen selbständig installieren und wieder deinstallieren müssen. Alle Testnutzer müssen die Installation innerhalb von 10 min und die Deinstallation innerhalb von 5 min fehlerfrei durchführen.

11.2.5 Empfehlungen zum Installationstest

Ein Problem des Installationstests kann die unbekannte Vielzahl der möglichen Systemvarianten sein. Daher ist es auch für den Installationstest sinnvoll, die wichtigsten Systemkonfigurationen[2] in einer Risikoanalyse zu priorisieren und die Zielumgebungen entsprechend der vorgenommenen Bewertung nachzustellen.

11.3 Zusammenfassung

- Code-Analysen und Installationstests stellen die Qualitätsmerkmale Änderbarkeit und Übertragbarkeit einer Software sicher.

[2] Siehe auch Interoperabilitätstest, Kap. 9.11.

- Mit Code-Analysen werden Dokumente, die Pseudo- oder Sourcecode enthalten, geprüft.
- Eine Code-Analyse kann von Experten ohne Werkzeuge in einem Code-Walkthrough, einer Code-Inspektion oder einem Schreibtischtest durchgeführt werden.
- Programmiersprachenspezifische Checklisten unterstützen Code-Inspektionen und Schreibtischtests.
- Der Schreibtischtest ist eine kostengünstige Alternative zu Code-Walkthrough und Code-Inspektion.
- Vor einer visuellen Code-Analyse wird automatisiert eine statische Code-Analyse mit einem Compiler oder einem speziellen Code Analyzer durchgeführt.
- Ein Installationstest muss durchgeführt werden, falls die Anwendung auf mehrere Rechner übertragen werden soll.
- Neben der Erstinstallation müssen auch die Installationen von Updates und die Deinstallation der Software getestet werden.
- Der Installationstest für Mobile-Apps muss Batteriestände und Störfalle berücksichtigen.
- Installationshandbücher werden einem Dokumententest unterzogen.
- Ein Usability-Test zum Installationsprozess kann wichtige Erkenntnisse bringen.

Tests zur Effizienz

12

Es ist nicht zu wenig Zeit, die wir haben, sondern es ist zu viel Zeit, die wir nicht nutzen.

Lucius Annaeus Seneca (1–65)

Die in diesem Kapitel beschriebenen Tests zur Effizienz sollen vermeiden, in die Schlagzeilen zu kommen:

> „Nach massiven Verkäufen ist in Tokio der Aktienhandel vorzeitig beendet worden – zum ersten Mal in der 56-jährigen Geschichte der Börse. …Die Zahl der Transaktionen hatte die Marke von vier Millionen erreicht und war damit an die Grenze der Kapazität des Computersystems der Börse gestoßen. …Es war das erste Mal in der 56-jährigen Geschichte der Tokioter Börse, dass der Handel vorzeitig eingestellt wurde. Allerdings gab es bereits Anfang November einen Computerausfall, der die Börse für viereinhalb Stunden lahm legte." (Zitat sueddeutsche.de)[1]

Ein Anwendungssystem muss sowohl unter normalen Betriebsbedingungen als auch in Ausnahmesituationen akzeptables Zeitverhalten aller Funktionen aufweisen und stabil bleiben. Diese Eigenschaften werden durch Effizienztests sichergestellt.

Ein Anwendungssystem muss sowohl unter normalen Betriebsbedingungen als auch in Ausnahmesituationen akzeptables Zeitverhalten aller Funktionen aufweisen und stabil bleiben. Diese Eigenschaften werden durch Effizienztests sichergestellt.

► Ein **Effizienztest** ermittelt die Effizienz[2] eines Softwareprodukts.

[1] Quelle: http://sz.de/1.819679. sueddeutsche.de, Süddeutsche Zeitung GmbH, 17.05.2010.

[2] Qualitätsmerkmal Effizienz, s. Abschn. 2.2.1.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_12

Diese Definition ist nicht gerade tiefschürfend, aber korrekt. Oft wird ganz allgemein von Performanz-/Lasttests[3] gesprochen, wenn eine Gruppe von verwandten Testarten gemeint ist. Dabei handelt es sich um Performanz-, Last-, Stress-, Massen-, Skalierbarkeits- und Speicherlecktest, die alle das Qualitätsmerkmal Effizienz überprüfen, allerdings mit unterschiedlichen Zielrichtungen und jeweils unter anderen Bedingungen.

Für unsere Testobjekte Webs(e)ite, Web-App und Mobile-App wird es in diesem Sinne immer spannend, wenn der Client (Desktop-PC oder Mobilgerät) mit dem Server kommuniziert.

Die Ausprägungen des Effizienztests werden in den folgenden Kapiteln beschrieben und versetzen die Tester in die Lage, folgende Fragen beantworten zu können:

- Werden Funktionen und Transaktionen[GL] innerhalb der geforderten Zeitvorgaben ausgeführt?
- Werden die geforderten Mengen in der geforderten Zeit übertragen und verarbeitet?
- Hält das System die Grundlast über längere Zeit?
- Können Belastungsspitzen des Systems ohne Qualitätsverlust abgefangen werden?
- Wird der Nutzer bei Überlastung informiert?
- Werden wohldefinierte Lastspitzen richtig ausbalanciert (Load Balancing)?
- Ist das System problemlos aufrüstbar?
- Tritt im Dauerbetrieb kein Performanzproblem durch Speicherlecks auf?
- Laufen keine Speicherbereiche voll und/oder über?
- Welchen Einfluss haben Netzwerkschwankungen auf die Performance?

12.1 Performanztest

Performanz (Zeitverhalten) beschreibt das Ausmaß, in dem eine Komponente oder ein System die geforderten Funktionen und Leistungen im Rahmen vorgegebener Bedingungen erbringt.

► Der **Performanztest** prüft den Ressourcenverbrauch (Zeit und Speicherplatz) eines Anwendungssystems unter normalen Systembedingungen.

Ein Performanztest wird in vier Schritten durchgeführt:

1. Schritt: Testszenarien pro Nutzergruppe festlegen
Im Performanztest wird ein realistischer Betrieb mit der zu testenden Anwendung simuliert. Die Testszenarien für den Performanztest bilden die Geschäftsprozesse ab, die von

[3] So wird auch im Folgenden oft „Performanz-/Lasttests“ geschrieben, auch wenn „Effizienztest“ korrekt wäre. Das Wort „Effizienztesttool“ klingt irgendwie nicht gut.

unterschiedlichen Nutzergruppen der Anwendung ausgeführt werden. Sie können mit Hilfe der anwendungsfallbasierten Testfallermittlung entworfen werden (s. Abschn. 4.1.5).

Nehmen wir unser Beispiel des Finanzierungsrechners wieder auf. Hier können für den Performanztest die Szenarien mit den drei Nutzergruppen Kunden, Interessenten und Informationssuchende aus dem Usability-Test (s. Kap. 10.3) übernommen werden:

- Kunden konfigurieren ihr Wunschauto und fordern ein verbindliches Finanzierungsangebot an.
- Interessenten konfigurieren mehrere Autos und berechnen dazu unterschiedliche Finanzierungsmöglichkeiten.
- Informationssuchende surfen durch die Anwendung und fordern allgemeine Finanzierungsunterlagen auf dem Postweg an.

2. Schritt: Anzahl, Zeitpunkt, Dauer und Frequenz festlegen
Um realistische Messergebnisse durch den Performanztest zu erhalten, wird für jedes Szenario aufgrund von Schätzungen, Umfragen oder Erfahrungswerten festlegt,

- wie viele Benutzer es parallel ausführen,
- zu welchen Zeitpunkten es ausgeführt wird,
- wie lange es durchgeführt wird und
- wie häufig es ausgeführt wird.

3. Schritt: Belastungsspitzen berücksichtigen
Bekannte, regelmäßig auftretende Situationen, die für eine bestimmte Dauer eine hohe Belastung für die Anwendung zur Folge haben, müssen beim Performanztest berücksichtigt werden.

Zum Beispiel muss bei einer Bankanwendung bedacht werden, dass sich täglich bei Börsenöffnung alle Aktienhändler gleichzeitig in die Broker-Software einloggen oder dass die Bank zu festen Terminen Massenüberweisungen veranlasst. Beides kann Auswirkungen auf die Performanz des Gesamtsystems haben. Oder ein Freemail-Anbieter muss damit rechnen, dass sich zwischen 12:00 und 13:00 Uhr, wenn in der Mittagspause privat gesurft wird, die Anzahl der Besucher seiner Website verdoppelt.

Solche Anwendungsspitzen treten im normalen Betrieb auf, sind planbar und müssen bei der Durchführung der Tests simuliert werden. Sie gehören nicht zu den Ausnahmesituationen, wie sie beim Lasttest (s. Kap. 12.2) betrachtet werden.

4. Schritt: Tests durchführen und auswerten
Die mit Anzahl, Zeitpunkt, Dauer und Frequenz versehenen Testszenarien werden in einem „Testdrehbuch“ zum Performanztest zusammengefasst und nach diesem durchgeführt.

Dabei werden unter anderem Antwortzeit- und Speicherverhalten des Systems gemessen und mit den Qualitätsanforderungen aus den Spezifikationen verglichen. Welche Größen im Performanztest gemessen und ausgewertet werden können, ist im Kap. 12.6 beschrieben.

12.2 Last- und Stresstest

Wenn die Ergebnisse der Performanzmessungen zufriedenstellend ausgefallen sind, müssen mögliche Ausnahmesituationen nachgestellt werden. Solche entstehen zum Beispiel, wenn aufgrund gestarteter Werbekampagnen oder besonderer Nachrichtensituationen ungewöhnlich viele Benutzer die Website aufsuchen.

► Der **Lasttest** prüft das Verhalten des Gesamtsystems bei steigender Systemlast.

Last wird für das zu testende System dadurch erzeugt, dass die Anzahl der parallelen Zugriffe auf das System stetig gesteigert wird. Zusätzlich kann auch die absolute Anzahl pro Zeiteinheit erhöht werden, in dem Pausen zwischen den Zugriffen verkürzt oder weggelassen werden. Das heißt, die Anzahl der Benutzer, die Wiederholungsraten der Transaktionen und die zu verarbeitenden Mengen werden so lange erhöht, bis das System abnormale Reaktionen zeigt.

Durch den Lasttest werden **Schwachstellen** (**Bottlenecks**) im System festgestellt. Gerade bei komplexen Webapplikationen sind die potentiellen Schwachstellen sehr vielfältig und nicht immer einfach zu lokalisieren. Engpässe können in einer komplexen Systemlandschaft überall auftreten, zum Beispiel im Webserver, im Applikationsserver, im DNS-Server[GL], im Router, in der Firewall, im Datenbank-Server oder in der Datenbank.

► Das System kann auch unter Last gesetzt werden, indem ihm große Datenmengen zur Verarbeitung zugeführt werden. Diese Art des Tests wird **Massentest** oder **Volumentest** genannt.

Zum Beispiel ist ein Massentest für eine Online-Anwendung notwendig, wenn temporär Systemprozesse zur Verarbeitung großer Datenmengen parallel auf dieselben Ressourcen wie die Online-Komponenten zugreifen müssen und die Anwendung rund um die Uhr zur Verfügung stehen muss.

Der geneigte Tester hat natürlich Spaß daran, nach dem Lasttest in einem Stresstest das Ganze bis zum Absturz des Systems zu treiben.

► Der **Stresstest** wird durchgeführt, um ein System an oder oberhalb der Grenzen, die in den Anforderungen spezifiziert wurden, bewerten zu können.

12.3 Skalierbarkeitstest

Während der Lasttest sein Augenmerk auf das Erkennen von Schwachstellen legt, wird mit dem Skalierbarkeitstest ermittelt, ob das System problemlos technisch aufgerüstet werden kann.

Skalierbarkeit ist die Fähigkeit eines Software-Produkts, technisch aufgerüstet werden zu können, um eine erhöhte Last zu verkraften.

Ein System skaliert zum Beispiel gut, wenn es bei der doppelten Anzahl von Prozessoren die Hälfte der Rechenzeit benötigt oder bei Verdoppelung der Belastung (Anzahl User, Anzahl Transaktionen) mit den doppelten Hardware-Ressourcen auskommt, ohne dass Eingriffe in die Software notwendig sind. Ein System skaliert schlecht, wenn bei doppelter Last die fünffachen Ressourcen zum Ausgleich der erhöhten Last benötigt werden.

► Der **Skalierbarkeitstest** prüft die Fähigkeit, ein Software-Produkt auf Rechnern unterschiedlicher Größe einsetzen zu können.

Beim Skalierbarkeitstest wird wie beim Lasttest die Belastung des Systems sukzessive gesteigert:

1. Stufe: Für jede Nutzergruppe werden die Testszenarien (Geschäftsvorfälle) einzeln durchgeführt.
2. Stufe: Die Szenarien werden pro Nutzergruppe zusammengefasst und parallel durchgeführt.
3. Stufe: Alle Szenarien aller Nutzergruppen werden gemeinsam durchgeführt.

12.4 Speicherlecktest

Ein **Speicherleck** (**memory leak**) ist ein Software-Fehler, bei dem ein Teil des Arbeitsspeichers nach Benutzung nicht wieder freigegeben wird. Über die Zeit verliert der betroffene Webserver verfügbaren Speicher, bis er „leckt", d. h. arbeitsunfähig ist oder sogar abstürzt.

► Der **Speicherlecktest** überprüft als ein auf Dauer angelegter Performanztest die Existenz von Speicherlecks.

Speicherlecks werden nach der Durchführung des Performanztests gesucht. Um sie aufzudecken, muss ein Performanztest gefahren werden, der vier bis fünf Tage dauert. Speicherlecks werden daran erkannt, dass Antwortzeiten oder Übertragungsraten stetig nachlassen.

Da schon „kleine“ Speicherlecks die Leistung eines Systems markant beeinflussen können, ist dieser Test auch für die Zuverlässigkeit eines Systems sehr wichtig.

Der Speicherlecktest im Rahmen des Performanztests wird allerdings keine Fehler im Speichermanagement eines einzelnen Programmes finden, sondern i.d. R nur die Symptome beim Gesamtsystem feststellen. Daher sollten im Unit-Test Code-Analysen durchgeführt und Profiler oder Memory-Checker (s. Abschn. 12.9.1) eingesetzt werden, um gezielte Speicherlecks zu eliminieren.

12.5 Automatisierung der Performanz-/Lasttests

Die beschriebenen Ausprägungen des Performanz-/Lasttests haben gemeinsam, dass sie dieselben Geschäftsprozesse simulieren. In der Testdurchführung unterscheiden sie sich allerdings in Dauer und Intensität. Es liegt auf der Hand, dass die Tests nicht manuell durchgeführt werden können, denn es müssten einige hundert oder sogar einige tausend Benutzer die Anwendung über einen längeren Zeitraum bedienen.

Performanz-/Lasttests werden mit **Lasttestwerkzeugen** (**Load Test Tools**) durchgeführt. Sie simulieren beliebig viele virtuelle Benutzer, welche die ihnen zugewiesenen Testszenarien parallel durchführen. Dabei werden Antwortzeiten aufgezeichnet, übertragene Datenmengen gezählt und verbrauchte Ressourcen (CPU, temporärer Speicher,…) gemessen.

Die Testszenarien werden i.d. R über die Browser-Oberfläche aufgezeichnet und anschließend auf Netzwerkprotokollebene wieder abgespielt. Damit die Skripte mehrfach und parallel eingesetzt werden können, werden die aufgezeichneten Testskripte parametrisiert. Wenn man mit dem Tool, mit dem funktionale Tests automatisiert wurden, „Glück“ hat, können die Testskripte aus dem Funktionstest für den Performanztest mit einem Lasttestwerkzeug genutzt werden, wie z. B. mit SoapUI und LoadUI (s. Abschn. 12.9.2.)

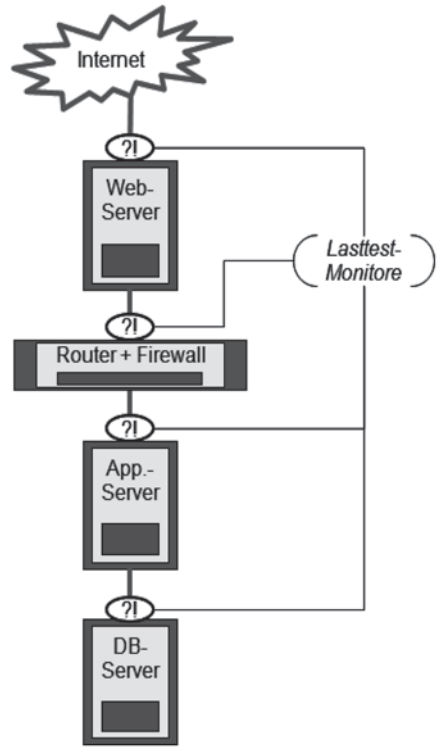
Mit **Monitoren** können Schwachstellen des Systems genau geortet werden. Sie können an unterschiedlichen Stellen im Netzwerk angelegt werden (s. Abb. 12.1). Monitore messen zum Beispiel die Transaktionszeiten zwischen Client und Server, die im Netzwerk auftretenden Verzögerungen, den Ressourcenverbrauch von Web- und Applikations-Servern und die Effizienz von Datenbankzugriffen.

12.6 Metriken der Performanz-/Lasttests

Im Folgenden werden Metriken zur Durchführung von Performanz-/Lasttests vorgestellt, die mittels Lasttestwerkzeugen gemessen und ausgewertet werden:

- **RPS** (**Requests Per Second**) – Anzahl der bearbeiteten Anfragen pro Sekunde: Sie zeigen den Umfang der Interaktionen zwischen Browser und Webserver. Wenn diese Größe mit steigender Last konstant bleibt, ist der Webserver am Limit.

Abb. 12.1 Monitore



- **CPU** – **Prozessorauslastung** des Servers in Prozent: Hierzu können Warnschwellen definiert werden, die dann auf eine zu hohe Auslastung hinweisen.
- **QR** (**Queued Requests**) – Anzahl der Abfragen in der Warteschlange: Wenn sie über eine längere Zeit größer Null ist, steht der Server unter Stress.

Beispiele für Performanzmessungen

Im folgenden Beispiel setzen wir den Webserver (und den Systemadministrator) unseres Finanzierungsrechners in zehn Testläufen langsam aber stetig unter Last. Als Testergebnis erhalten wir drei Messkurven zu den Metriken RPS, CPU und QR.

1. **Beispiel: Requests Per Second**

Die in Abb. 12.2 dargestellte Messkurve zeigt die Anzahl der bearbeiteten Anfragen an den Webserver pro Sekunde (RPS) bei steigender Anzahl der Threads[GL]. Die Anzahl der geladenen Threads steigt von 1 auf 250, wobei die RPS maximal auf 41 steigen.

Erkenntnis: Vom System können nicht mehr Requests verarbeitet werden, es ist an seine Grenzen gestoßen.

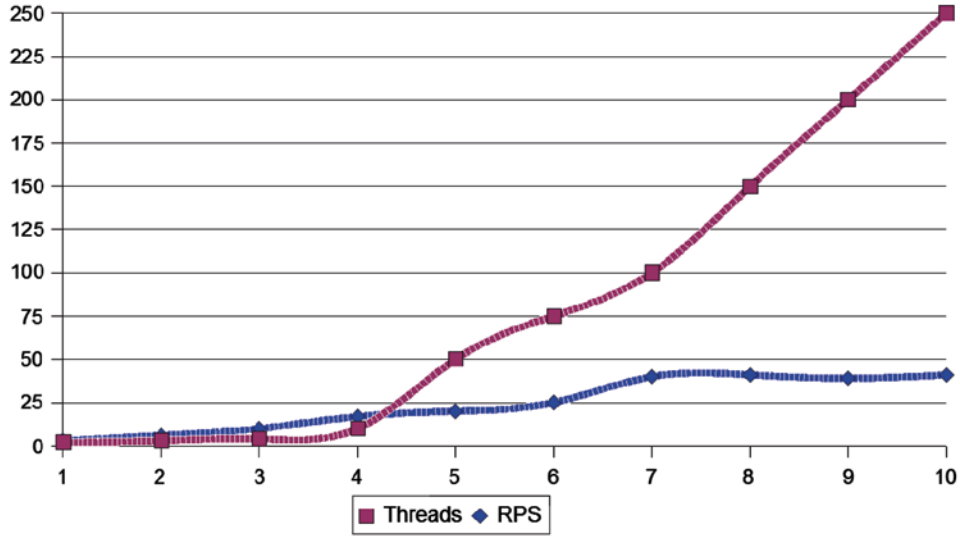


Abb. 12.2 Messung RPS

2. **Beispiel: Prozessorauslastung**
Die Messkurve in Abb. 12.3 zeigt, dass der Prozessor des Webservers keine Schwachstelle darstellt, denn seine Auslastung liegt bei steigender Anzahl der Threads bei maximal 55 %.

Erkenntnis: In diesem Aspekt skaliert das System gut.

3. **Beispiel: Queued Requests**
Die dritte Messreihe in Abb. 12.4 zeigt die Anzahl der Queued Requests (QR) in Abhängigkeit der Thread-Anzahl. Der Anstieg der Queued Requests auf fast 200 bei diesem Testszenario ist extrem.

Erkenntnis: Eine Kommunikationskomponente des Webservers stellt eine Schwachstelle dar.

Es gibt weitere aussagekräftige Metriken zum Performanz-/Lasttest:

- Eine **Transaktion** umfasst die Anfrage (Request) an den Server, die Bearbeitung der Anfrage und die Antwort (Response) vom Server. Hier ist interessant zu wissen, ob die **Anzahl abgeschlossener Transaktionen** pro Sekunde proportional zur Steigerung der Anfragen bleibt.
- **Transaktionsdauer**: Unterschiedliche Testszenarien lösen unterschiedliche Transaktionen aus. Stellt beim Testen von mehreren Transaktionen eine Transaktion einen zeitlichen Ausreißer dar, so deutet das auf Fehler in der Programmierung hin.
- Die **Umlaufzeit (Round Trip Time)** misst die Dauer zwischen Beginn und Abschluss einer Transaktion und sollte unter Stress konstant bleiben. Werden die Transaktionen

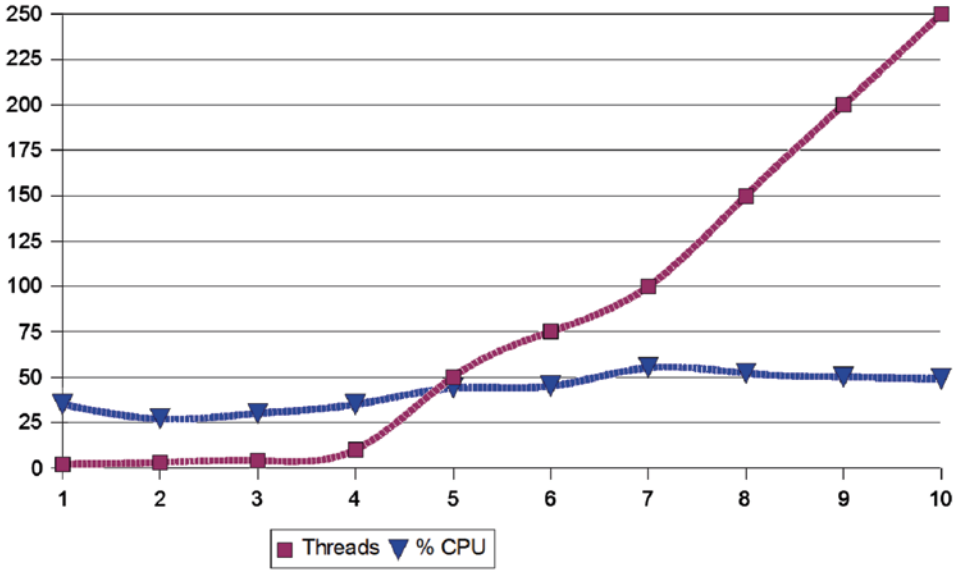


Abb. 12.3 Messung CPU-Auslastung

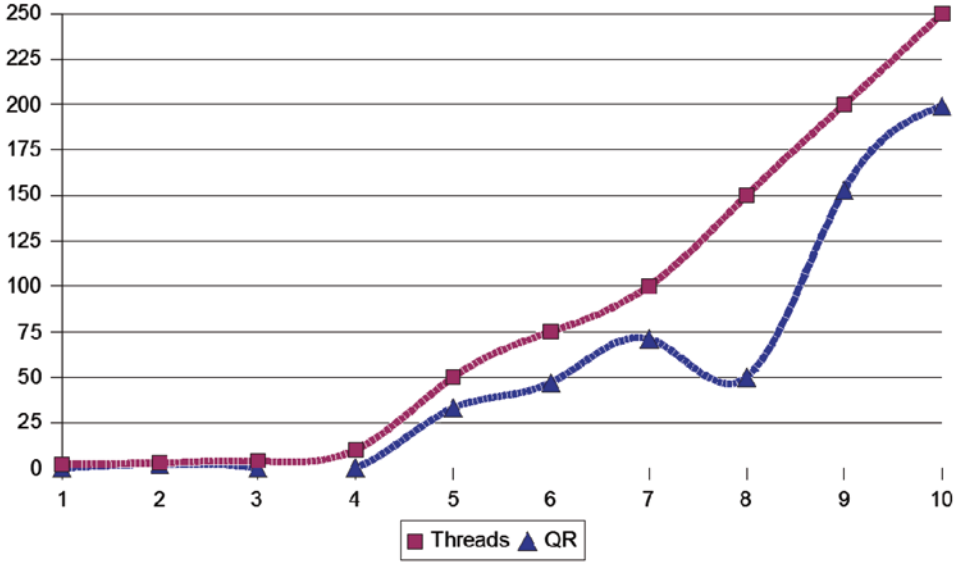


Abb. 12.4 Messung QR

generell langsamer und misslingen, kann der Server keine weiteren Anfragen mehr bearbeiten.

- Die **Latenzzeit** (kurz Latenz) gibt die Zeit an, die ein Paket in einem Netzwerk von einem Punkt zu einem anderen und wieder zurück braucht, zum Beispiel vom Client zum Server und wieder zurück. Die Latenz entspricht in etwa der Hälfte der Umlaufzeit (Hin- und Rückweg).

  Zu große Latenzzeiten können mit persistenten Verbindungen zum Server reduziert werden, d. h. nicht für jede Anfrage wird eine eigene Verbindung hergestellt. Mit dem HTTP-Protokoll 1.1 können sogar zwei persistente Verbindungen parallel aufgebaut werde. Diese Lösungen können sich aber wiederum aufgrund der Anzahl der offenen Verbindungen schlecht auf die Performanz auswirken. Einen Tod muss man sterben.
- **Anzahl offener Verbindungen**: Steigt die Anzahl der gleichzeitigen Verbindungen zum Webserver bei gleich bleibender Anzahl von Anfragen, bedeutet das, dass die Verbindungen länger als notwendig offen bleiben.
- Der verfügbare **Speicherplatz** steigt mit jeder neuen Verbindung und muss nach Beendigung der Verbindung wieder freigegeben werden. Verringert sich bei gleichbleibender Anzahl von wechselnden Verbindungen die Größe des verfügbaren Speichers stetig, so ist das ein Hinweis auf ein Speicherleck.
- Der **Datendurchsatz** misst die Menge an übertragenen Daten vom und zum Webserver in Kilobyte pro Sekunde. Ist die Anzahl Kilobyte nicht mehr proportional zur Anzahl der Nutzer, hat das System seine Grenzen erreicht.

Beispiel: Datendurchsatz

Eine typische Messung des Datendurchsatzes ist in Abb. 12.5 dargestellt. Sie zeigt, dass ab 25 Usern der Datendurchsatz im System rapide abnimmt.

Erkenntnis: In diesem Fall hat das Load Balancing[4] nicht richtig gearbeitet.

Es geht noch weiter mit messbaren Zeiten, die schon beim Systemdesign im Auge behalten werden müssen:

- Die **DNS-Time** ist die Zeit, die der DNS-Server[GL] benötigt, um den URL-Namen in seine IP-Adresse aufzulösen und an das anfragende System zurückzugeben. Diese Zeit ist Null, wenn die IP-Adresse in der Hosts-Datei[GL] auf dem Client des Benutzers oder im Cache des Webbrowsers abgelegt ist.
- Die **First-Byte-Time** ist die Zeit, die der Server braucht, um das erste Daten-Byte der Anfrage auszuliefern. Sie ist zum Beispiel bei statischen HTML-Seiten geringer als bei dynamisch generierten.
- Wie schnell sieht der Benutzer den Inhalt einer Website? Die **Content-Time** (Ladezeit) ist die Dauer, in der die gesamten Dateninhalte einer Anfrage an den Browser übertra-

[4] Zu Load Balancing s. Abschn. 13.1.1.

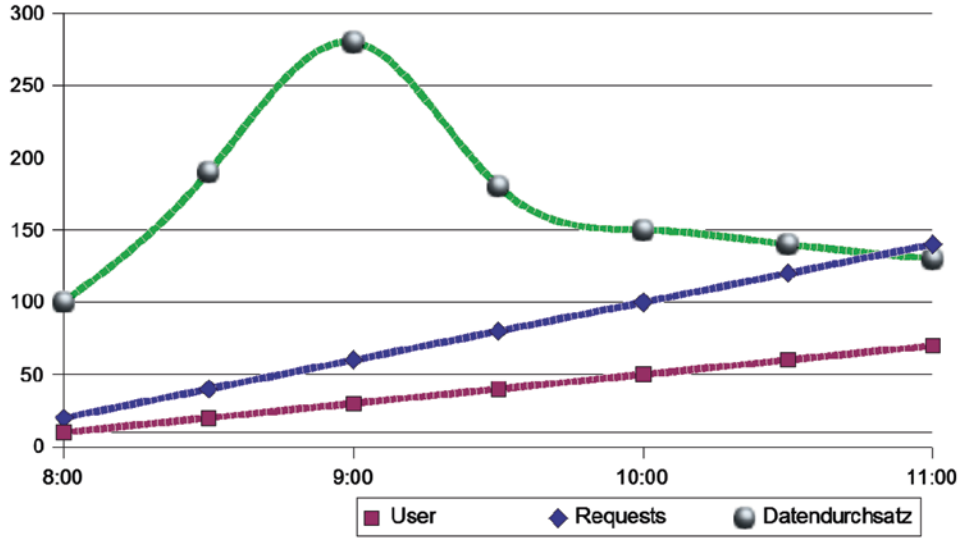


Abb. 12.5 Messung Datendurchsatz

gen werden. Sie hängt zum Beispiel von der Bandbreite der Verbindung, der Größe der Datei, und davon ab, ob diese Datei statisch ist oder dynamisch erzeugt wird.

- Die **Connection-Time** ist die Dauer der Verbindungsherstellung zwischen Browser und Webserver, d. h. die Zeit zwischen einer Anfrage an den Webserver bis zum Erreichen der Bestätigung im Client. Die Connection-Time kann durch persistente und parallele Verbindungen optimiert werden. Damit die Anzahl der offenen Verbindungen nicht zu groß wird, muss im Webserver die Verbindungsdauer und die Anzahl der erlaubten parallelen Verbindungen „vernünftig" konfiguriert werden. Was „vernünftig" ist, wird durch die Performanz- und Lasttestmessungen festgestellt.

Die Erkenntnisse über diese einzelnen Metriken liefern eine konstruktive Maßnahme in Form der Checkliste „Performanz Webseite":

✓Checkliste Performanz Webseite

Optimierung

- Wird der Content der Webseite sukzessive geladen, so dass der Benutzer nicht auf den komplett zu ladenden Inhalt warten muss?
- Werden möglichst nur statische Dateien übertragen?
- Wurde die optimale Dauer der Verbindungen bestimmt und richtig konfiguriert?
- Werden Verbindungen nicht unnötig lange offen gehalten?
- Sind, wenn technisch möglich, Requests gebündelt, damit nicht permanent Verbindungen auf und zugemacht werden müssen?
- Wurde die optimale Anzahl der parallelen Verbindungen bestimmt und richtig konfiguriert?

- Kann der DNS-Server umgangen werden und wird das Verfahren ggf. eingesetzt?
- Sind externe JavaScript-Dateien am Ende der HTML-Dateien eingebunden, weil sie parallele Downloads verhindern?
- Werden Grafiken möglichst ohne CSS dargestellt?
- Werden Bilder und Grafiken im HTML-Code nicht skaliert?
- Gibt es keine „leeren“ Bilder im Code, wie z. B. <img src = """>?

Komprimierung
- Werden die zu übertragenden Dateien komprimiert (HTML, CSS, JavaScript)?
- Sind alle Bilder und Grafiken so komprimiert, dass sie schnell geladen werden?

Cache
- Sind JavaScripts und CSS-Definitionen in externe Dateien ausgelagert, damit sie im Cache gespeichert werden?
- Werden Grafiken, die nicht oder selten geändert werden, im Browsercache gehalten?
- Werden Grafiken, die oft geändert werden, nur kurz oder gar nicht im Browsercache gehalten?
- Sind die Webseiten-Komponenten maximal 25 KB groß, damit sie (z. B. im iPhone) gecacht werden?
- Wird das Caching auf dem Webserver kontrolliert

Wer noch weitere Möglichkeiten sucht, eine Website schnell zu machen, wird bei Yahoo! [URL: Yahoo-Regeln] fündig. Diese Regeln können zum Beispiel online mit YSlow [URL: YSlow] überprüft werden. YSlow ist im Abschn. 12.9.1 beschrieben.

12.7 Mobile-Performanz-/Lasttest

► Der Mobile-Performanz-/Lasttest testet nicht die Performanz des Endgeräts und auch nicht die Performanz der Mobile-Appauf dem Endgerät, sondern das Back-End zusammen mit den Kommunikationsverbindungen zum Front-End.[5]

Beim Performanztest im mobilen Umfeld müssen neben den Punkten aus Kap. 12.6 weitere Aspekte berücksichtigt werden.

Umlaufzeiten optimieren
Die Dauer eines Round Trips, d. h. die Dauer zwischen Beginn und Abschluss einer Transaktion, hängt bei mobilen Geräten vom Ladezustand der Batterie und von der Qualität der Netzverbindung ab. Und eine hohe Anzahl von Round Trips zehrt am Akku und an der La-

[5] Zu Testobjekten im Front-End und Back-End s. Kap. 2.1.

tenz. Daher müssen die Umlaufzeiten für Mobile-Webseiten und für Mobile-Apps unbedingt optimiert sein, was mit der obigen Checkliste „Performanz Webseite“ geprüft wird.

Zum Beispiel werden bei einer optimierten Suchfunktion die Eingaben vom Benutzer nicht zeichenweise „just in time“ an den Server übergeben, sondern erst wenn der Benutzer seine Eingabe explizit absendet.

Unterschiedliche Netze testen
Das mobile Gerät kann via Wi-Fi (WLAN) oder via Mobilfunknetz (GSM, LTE, UMTS,…) mit dem Server verbunden sein; jede Verbindungsart hat eine andere Bandbreite. Und ein Smartphone kann aus technischen Gründen nicht die ganze Bandbreite eines WLANs ausnutzen. Unterschiedliche Mobilfunknetze haben auch unterschiedliche Latenzzeiten. Diese Aspekte müssen in den Performanztest eingehen, denn schlechte Netzqualität bringt hohe Last und somit schlechte Performanz.

Testobjekte in Bewegung halten
Da sich mobile Geräte per Definition in Bewegung befinden können, ist eine Funkverbindung nicht immer konstant. Der Empfang kann mal besser, mal schlechter sein, mal kann er im Funkloch abreißen. Datenpakete können somit fehlerhaft sein oder gar verloren gehen und müssen nochmal übertragen werden.

Rail Driven Testing[6]
Ein realistischer Systemtest für Mobile-Websites und Mobile-Apps ist also der Kauf einer Bahnfahrkarte für einen Zug ohne Hotspot und die Testdurchführung auf einer Bahnstrecke mit Tunneln und einer Endstation mit schlechter Funkverbindung.

Netzwerkeinflüsse berücksichtigen
Alle diese und weitere Faktoren können sich auf die Performanz beim Laden einer Website, des Downloads einer Mobile-App oder die Kommunikation mit dem Webserver auswirken. Sie müssen beim Performanz-/Lasttest berücksichtigt und getestet werden. Zum Beispiel können in dem Tool NeoLoad Lastprofile mit unterschiedlich guter bzw. schlechter Netzverbindung für die Testdurchführung festgelegt werden (s. Abschn. 12.9.2).

Die Checkliste „Mobile-Performanztest“ gibt vor, unter welchen Systembedingungen die fachlichen Testszenarien, die für den Performanztest definiert wurden und die eine Kommunikation mit dem Back-End erfordern, ausgeführt und überprüft werden müssen:

✓ Checkliste Mobile-Performanztest
Teste die Anwendung
- mit unterschiedlichen Verbindungsarten (EDGE, GRPS, HSPA, LTE, UMTS, WLAN, Satellit),
- mit unterschiedlich schnellen Verbindungen (langsam, schnell mittelmäßig),

[6] Rail Driven Testing ist noch(!) kein offizieller Testbegriff. ☺

- bei sehr niedrigem Akkustand,
- bei schlechter Funknetzverbindung,
- bei schlechter WLAN-Verbindung,
- mit unterschiedlichen Netzkonfigurationen im Inland (Wechsel der SIM-Karte),
- mit unterschiedlichen Netzkonfigurationen im Ausland (Wechsel der SIM-Karte),
- während Funkzellen beim eigenen Netzanbieter wechseln (Roaming[GL]),
- bei grenzüberschreitendem Roaming[GL] mit anderen Anbietern.

Sicherlich wäre es schön, wenn hier alle möglichen Kombinationen getestet werden könnten (Gerätetyp, Verbindungsart, Bandbreite, Verbindungsqualität, SIM-Karten, Roaming-Arten,…), geht aber nicht. Um die Anzahl der Konstellationen für den Performanztest halbwegs realistisch und bezahlbar zu halten, sei hier wieder einmal auf die Methoden des Paarweisen Testens (s. Kap. 5) hingewiesen.

12.8 Planung der Effizienztests

Produktionsnah testen
Effizienztests müssen in einer separaten, produktionsnahen Systemumgebung durchgeführt werden, damit die Testergebnisse nicht von äußeren Einflüssen verfälscht werden und wiederholte Testläufe vergleichbar sind.

Es ist zum Beispiel schon vorgekommen, dass eine Anwendung durch falsche Zuweisung von IP-Adressen nur im internen Netz getestet wurde. Die Firewall, die sich später nach der Freigabe des Systems als Schwachstelle herausstellte, war unerkannter Weise nicht in den Test miteinbezogen worden.

Technische Ausstattung prüfen
Ein weiterer kritischer Punkt für den Performanztest ist die technische Ausstattung des Testnetzes. Es wird ein Verbund von Test-Clients benötigt, auf denen virtuelle User ihre Tests (d. h. die Testskripte) ausführen. Jeder virtuelle User verbraucht je nach eingesetztem Testwerkzeug nicht wenig Speicherplatz. Hier muss bei Auswahl des Load Test Tools genau geprüft werden, ob die zur Verfügung stehende Kapazität ausreicht.

Dazu ein einfaches Beispiel: Wenn auf einem Client maximal 50 virtuelle User simuliert werden können, werden für einen Lasttest mit 2000 Usern 40 Rechner als Lastgeneratoren benötigt. Der Performanztest muss also frühzeitig geplant werden. Unter Umständen kann es sich rechnen, für die Dauer des Tests eine Test-Server-Farm anzumieten.

Die bisherigen Ausführungen zu den Performanz-/Lasttests münden in einer Checkliste, die zur Planung und zur Durchführung der Effizienztests zu Rate gezogen werden sollte:

✓ Checkliste Planung Effizienztest

Testziele und -szenarien

- Sind die Testziele definiert (Performanz, Last, Skalierbarkeit, Speicherleck, Stress, Volumen)?
- Sind die Qualitätsanforderungen definiert (durchschnittliche Transaktionszeit, maximale Download-Zeit, durchschnittliche Prozessorauslastung,…)?
- Sind die Testszenarien und Testskripte den Testzielen entsprechend festgelegt?
- Beruhen die Testszenarien auf realistischen Geschäftsprozessen?
- Sind die trotz Automatisierung notwendigen manuellen Testschritte festgelegt?
- Gibt es eine konkrete Planung für die Testdurchführung, d. h. ein detailliertes Testdrehbuch?

Technische Voraussetzungen

- Ist die Plattform der zu messenden Hardware bekannt?
- Sind alle technischen Komponenten der Testumgebung bekannt?
- Sind alle technischen Daten der Netzwerkumgebung bekannt?
- Sind alle Adressen der Netzwerkumgebung bekannt?
- Ist die Netzwerkkonfiguration für die geplanten Tests ausreichend?
- Spiegelt die Testumgebung die Produktionsumgebung hinreichend wieder oder besser: wird in einer Kopie der Produktionsumgebung getestet?
- Ist die Hardware-Ausstattung der Testrechner (Server, Clients) für die Tests ausreichend?
- Können die Testziele mit dem beschafften/existierenden Load Test Tool erreicht werden (Plattform- und Netzwerkkompatibilität, Speicherverbrauch)?
- Können alle verwendeten Webtechniken in den Lasttest mit einfließen wie zum Beispiel AIR[GL], ASP.NET[GL], Flex[GL], JavaFX[GL], Silverlight[GL], RTMP[GL]?
- Sind die Messpunkte definiert, um die geforderten Messwerte zu erhalten?
- Sind die notwendigen Monitore vorhanden, um alle definierten Messpunkte abgreifen zu können?
- Sind die Monitore richtig installiert und parametrisiert?

Testdurchführung

- Hat die Anwendung vor dem Performanztest alle anderen Prüfungen und Tests erfolgreich bestanden?
- Stehen die benötigten Testdaten zur Verfügung?
- Sind alle für die Tests notwendigen Berechtigungen eingerichtet?
- Ist die exklusive Verfügbarkeit der Testumgebung über die gesamte Testdauer sichergestellt?
- Ist sichergestellt, dass die Testdurchführung nicht durch Fremdeinwirkung verfälscht wird?
- Sind länder- und zeitzonenübergreifende Performanz- und Lasttests geplant?

Mobiles Umfeld

- Ist festgelegt, ob und welche Simulatoren/Emulatoren für mobile Geräte eingesetzt werden können bzw. sollen?
- Stehen alle notwendigen mobilen Endgeräte zur richtigen Zeit real zur Verfügung (intern, extern, in einer Cloud)?
- Kann das Lasttesttool Tests aufzeichnen, die auf mobilen Geräten ausgeführt werden, insbesondere mit der Hand ausgeführte Aktionen (z. B. Wischen)?
- Kann das Lasttest-Tool mobile Endgeräte simulieren, damit nicht ungewollt auf Desktop- bzw. Mobile-Seiten umgeleitet wird und die falschen Testobjekten getestet werden?
- Kann das Lasttest-Tool Browser, Bandbreiten, Netzwerke und Verbindungsqualitäten simulieren?

12.9 Werkzeuge für Effizienztests

Effizienztests müssen „im Großen“ auf Systemebene durchgeführt werden, weil das Zusammenspiel aller Systemkomponenten untersucht wird.

Aber um Fehler zu vermeiden bzw. möglichst früh zu finden, können einzelne Testobjekt wie z. B. Webseiten oder Datenbankzugriffsmodule vorab „im Kleinen“ auf ihre Performanzeigenschaften geprüft werden.

12.9.1 Effizienztests im Kleinen

Um die Performanzeigenschaften einer einzelnen Webseite festzustellen, gibt es frei verfügbare Tools wie **PageSpeed** von Google [URL: PageSpeed] oder **YSlow** von Yahoo! [URL: YSlow]. Sie können online oder als Add-on in Browsern ausgeführt werden.

Mit diesen Tools werden keine realen Zeiten gemessen, sondern mit einem anpassbaren Regelwerk werden die **Einflussfaktoren der Performanz** für eine einzelne Webseite analysiert und bewertet. Als Ergebnis erhält die Webseite eine Bewertungszahl x von 100 möglichen Punkten. Kennen die Entwickler das Regelwerk, so ist das eine effektive konstruktive QS-Maßnahme.

Das schon im Kap. 12.7 erwähnte YSlow [URL: YSlow] läuft als Add-on in mehreren Browsern, wie zum Beispiel in Firefox, wobei das Entwicklerwerkzeug Firebug benötigt wird. In YSlow können unterschiedliche Regelsätze eingestellt werden. In Abb. 12.6 ist zum Beispiel eine Webseite mit dem „Klassischen Regelsatz“ durch YSlow geprüft worden; sie hat 60 von 100 Performanz-Punkten und einen schlechten Bewertungsgrad „E“ erreicht.

Sollen schon im Rahmen der Unit-Tests für einzelne Module Aussagen zur Performanz getroffen werden oder müssen aufgrund der Testergebnisse der Performanz-/Lasttests einzelne Module untersucht werden, kommen Profiler und Memory-Checker zum Einsatz.

Mit ihnen wird der Zeit- und Speicherplatzverbrauch auf Code-Ebene überprüft, was mit Lasttest-Tools nicht möglich ist. Java Profiler sind zum Beispiel unter [URL: TestToolsJava] in der Rubrik „Profilers“ zu finden.

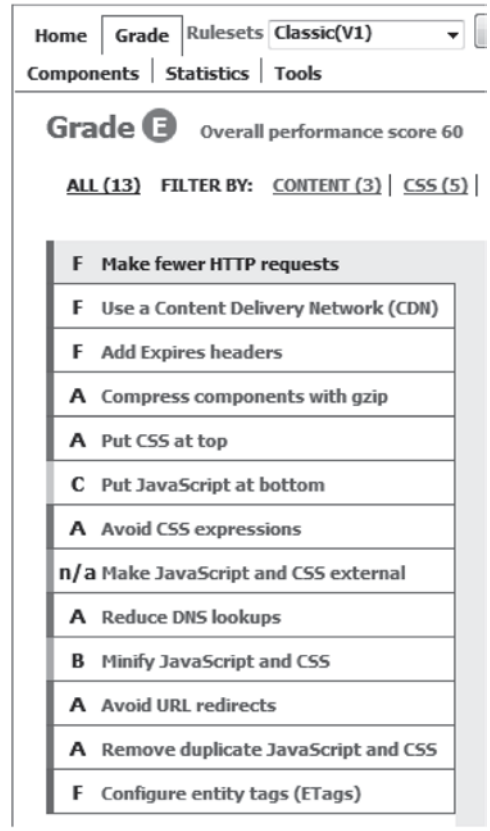


Abb. 12.6 Performanzanalyse mit YSlow

12.9.2 Effizienztests im Großen

Die Arbeitsweise von **Lasttestwerkzeugen** (**Load Test Tools**) und Monitoren ist bereits im Kap. 12.5 beschrieben. Es gibt eine ganze Reihe von freien und kommerziellen Lasttestwerkzeugen (s. [URL: TestToolsOPENSOURCE] Rubrik „Testing tools – Performance testing“ und [URL: TestToolsSQA] Rubrik „Web Tools – Load and Performance Test Tools“). Frei verfügbare Tools können je nach Zielsetzung und Wichtigkeit der Webapplikation eine Alternative sein, auch wenn sie nicht immer einen so großen Funktionsumfang besitzen und so komfortabel sind wie die kommerziellen Tools. Einige Tools werden im Folgenden vorgestellt.

Eines der bekanntesten, frei verfügbaren Werkzeuge ist das von der Apache Jakarta Group entwickelte **JMeter** [URL: JMeter]. Es ist in Java geschrieben und steht als Open Source unter Apache Lizenz. JMeter ist besonders für die Performanzmessungen von HTTP-Servern geeignet. Übertragene Texte können auch inhaltlich überprüft werden.

Mit sogenannten Samplern wird ein Server unter Last gesetzt. Mit einem HTTP Request wird zum Beispiel ein Webserver unter Last gesetzt, mit einem FTP Request ein FTP-Server und mit einem JDBC Request eine Datenbank. Unter anderem stehen Sampler

für HTTP-, FTP-, JDBC-, Java-, SOAP/XML-RPC-, SOAP-, LDAP-Requests und JUnit zur Verfügung.

Wie bei vielen Open Source Tools üblich, ist JMeter durch ein Plugin-Konzept jederzeit erweiterbar; dieses gilt sowohl für die Sampler als auch für die Monitore. Im Internet finden sich auch kommerzielle Anbieter, die solche Module anbieten.

Ein weiteres Open Source Tool für Webservices und Apps ist **LoadUI** [URL: LoadUI] mit dem interaktiv per Drag & Drop real-time Lasttestszenarien zusammengestellt und durchgeführt werden können.

Funktionale Testszenarien, die mit SoapUI (s. Abschn. 9.9.1) erstellt wurden, können in LoadUI integriert werden, wie in Abb. 12.7 mit den Webservice-Testfällen unseres Währungsrechners dargestellt. In diesem sehr einfachen Szenario wurde 15 Minuten lang ein

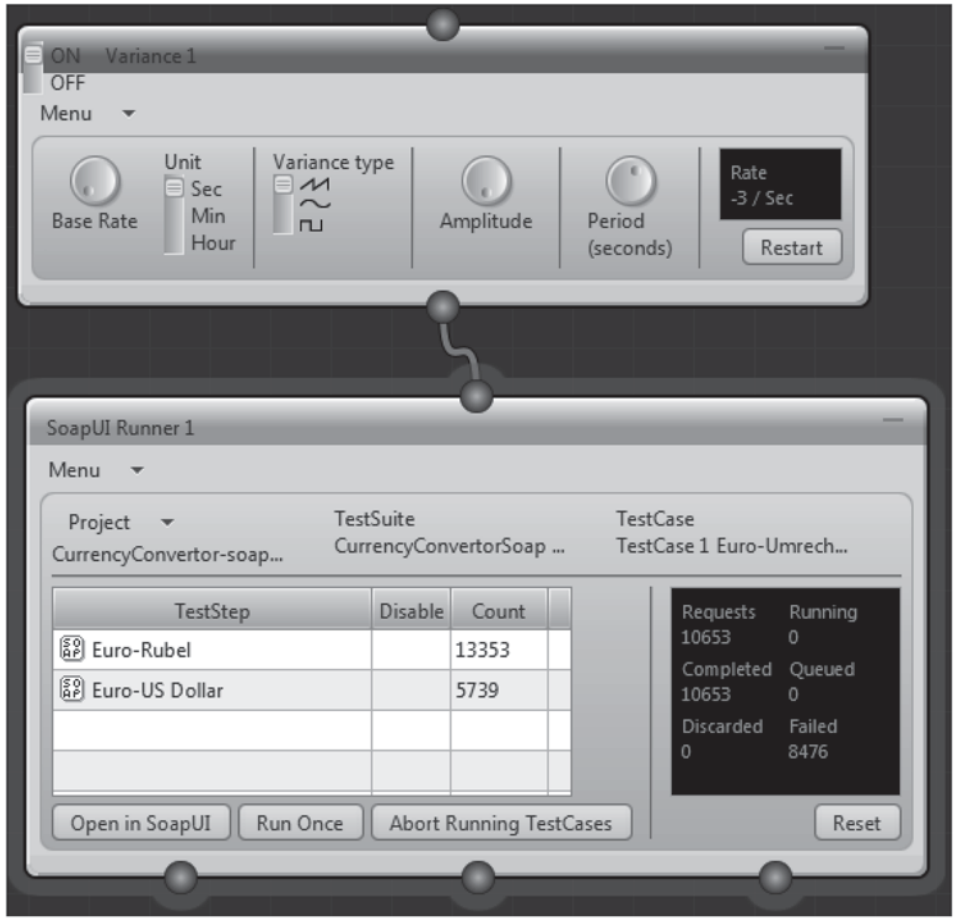


Abb. 12.7 Szenario LoadUI

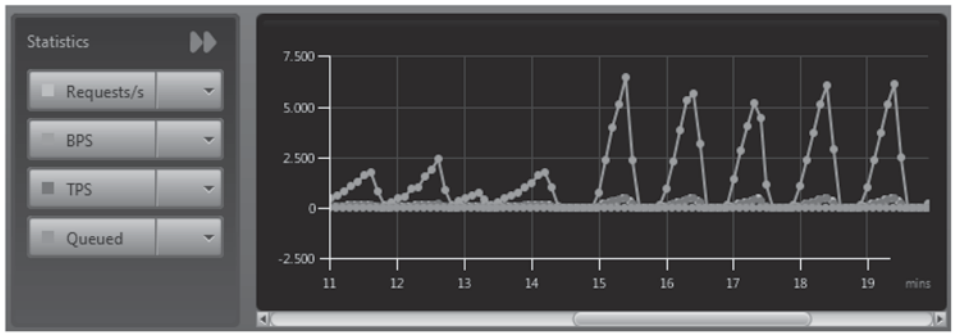


Abb. 12.8 Lasttest LoadUI

Performanztest mit bis zu 20 Usern durchgeführt. Anschließend wurde im Minutentakt mit bis zu 100 Usern Last erzeugt. Das Ergebnis in Abb. 12.8 zeigt keine Anomalien.

Ein kommerzielles Tool, dass sich auf Web- und Mobile-Technologien spezialisiert hat, ist **NeoLoad** [URL: NeoLoad]. Interessant bei NeoLoad ist, dass die virtuellen User für gewisse Zeiten gemietet werden können und dass via Cloud-Service ein geografisch verteilter Lasttest durchgeführt werden kann.

Wie im Kap. 12.7 beschrieben, spielt im mobilen Bereich die Netzwerkverbindung eine gravierende Rolle für die Performanz. Für den Performanz-/Lasttest heißt das, dass Verbindungen mit unterschiedlicher Bandbreite und Verbindungsqualität getestet werden müssen. In NeoLoad zum Beispiel können den Testszenarien realitätsnah unterschiedliche Mobilfunknetze, einschließlich Bandbreitenbeschränkungen, Latenzzeit und Paketverlust zugewiesen werden, s. Abb. 12.9.

NeoLoad bietet zudem die Möglichkeit, einen Vergleich zweier Testläufe grafisch aufzubereiten. Diese Funktionalität ist in größeren Projekten von Interesse, um bewerten zu können, ob sich durchgeführte Optimierungsmaßnahmen tatsächlich positiv auf die Performanz des Systems auswirken.

12.10 Qualitätsanforderungen zum Effizienztest

Auftraggeber von Software-Anwendungen geben nicht immer konkrete und messbare Werte zum Qualitätsmerkmal Effizienz vor. Die Folge sind undankbare Diskussionen bei der Abnahme, ob das Zeitverhalten zumutbar oder inakzeptabel ist. Um das zu vermeiden, muss der Auftraggeber im Rahmen der Spezifikation der nicht funktionalen Anforderungen konkrete Vorgaben zu den im Kap. 12.6 beschriebenen Metriken machen, wie zum Beispiel:

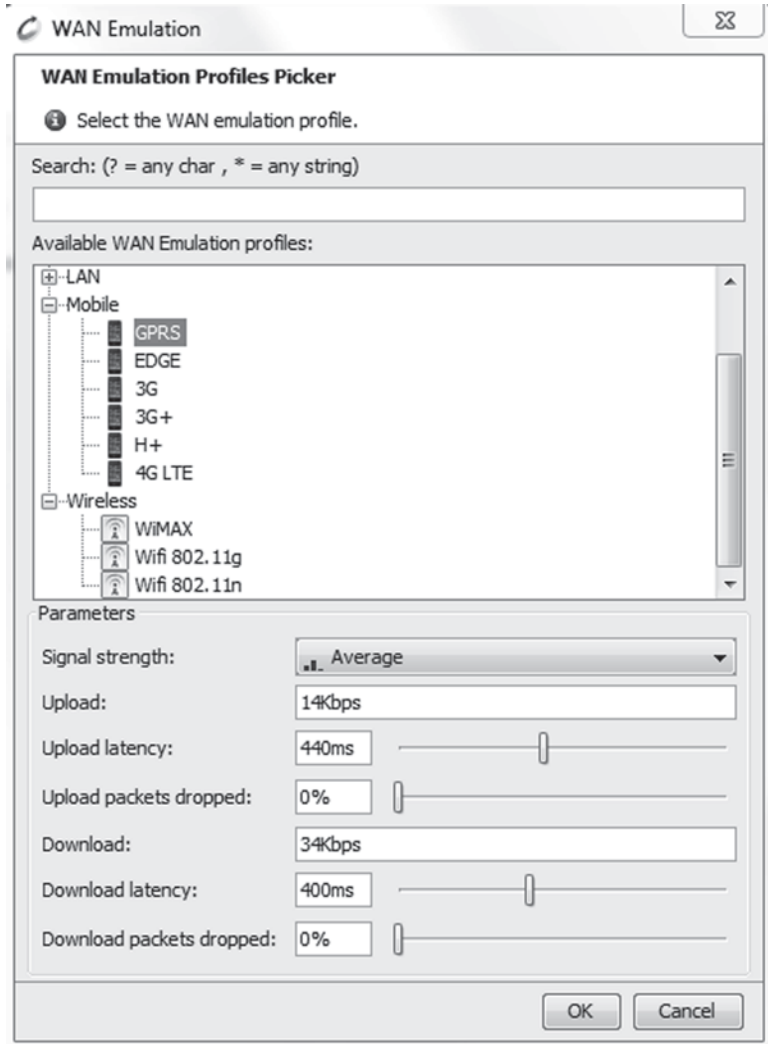


Abb. 12.9 Netzeinstellungen in NeoLoad

- Die Transaktion zur verbindlichen Bestätigung einer Finanzierungsanfrage muss in 5 Sekunden abgeschlossen sein.
- Der Download eines 500 KB großen PDF-Dokuments muss bei einer 3000er DSL-Verbindung in 3 Sekunden abgeschlossen sein.
- Bei Spitzen von 300 gleichzeitig angemeldeten Benutzern darf sich die Performanz des Systems höchstens um 20 % verschlechtern (Antwortzeiten, Downloadzeiten,…). Dabei wird von einer maximalen Anzahl von 150 gleichzeitig angemeldeten Benutzern ausgegangen.

- Der Speicherlecktest mit einer Dauer von 5 Tagen zeigt keine Verschlechterung in der durchschnittlichen Transaktionsdauer. Vorrausetzung ist, dass folgende Szenarien über den Tag verteilt durchgeführt werden: 2000 Benutzer suchen nach Informationen, 500 laden Dokumente herunter, 200 fordern Informationsmaterial an, 50 berechnen Leasingraten und 10 schließen online einen verbindlichen Leasingvertrag ab.
- Auf dem Client muss das Laden eines Java-Applets innerhalb von 6 Sekunden abgeschlossen sein.

Natürlich müssen die konkreten Systembedingungen, unter denen diese Qualitätsanforderungen nachgewiesen werden sollen, festgelegt werden. Dazu gehören zum Beispiel die Prozessorleistungen der Rechner (Clients und Server), die Anzahl der parallel geschalteten Prozessoren oder die Bandbreiten zur Datenübertragung.

Für Webseiten können Regelwerke für Performanzeigenschaften festgelegt und der Erreichungsgrad vorgegeben werden. Zum Beispiel müssen alle Webseiten nach dem Yahoo!-Regelsatz „Classic(v1)“ mindestens den Grad „C“ erreichen und kein Prüfpunkt darf den Grad „E“ oder „F“ erhalten (bei einer Skala von A bis F).

12.11 Empfehlungen zum Effizienztest

Performanztests in allen Phasen
Die Performanz sollte schon bei der Entwicklung und Integration der einzelnen Komponenten eine Rolle spielen. Wenn möglich sollten in frühen Projektphasen erste Aussagen zum Laufzeitverhalten gemacht werden, um die Ergebnisse auf das Gesamtsystem hochrechnen zu können. Dann können Performanzprobleme, die auf einer nicht optimalen Programmierung einzelner Komponenten beruhen, frühzeitig beseitigt werden.

Messungen, die endgültige Aussagen über das Verhalten des Gesamtsystems erlauben, können erst in den späteren Teststufen Systemtest und Abnahmetest durchgeführt werden. Im laufenden Betrieb muss die Performanz permanent überwacht werden, um frühzeitig einen Leistungsabfall des Systems zu erkennen.

Skalierbarkeit prüfen
Wenn ein System schlecht skaliert oder die Performanz eines Systems unzureichend ist, ist der Zukauf von Prozessorleistung ein beliebter Lösungsversuch, der i.d. R jedoch nur begrenzten Erfolg verspricht. Ein Skalierbarkeitstest in einer frühen Testphase schützt vor unliebsamen Überraschungen, wenn das System aufgerüstet werden muss.

BestehendeTestszenarien verwenden
Testszenarien für den Performanz-/Lasttest müssen nicht immer neu „erfunden“ werden. Geeignete Kandidaten sind nicht zu kompliziert Testszenarien aus dem funktionalen Systemtest oder Anwendungsfälle aus dem Usability-Test, wie im Beispiel des Fi-

nanzzierungsrechners mit den nach Nutzergruppen unterschiedenen Aufgabenstellungen (s. Abschn. 10.3.1).

Länderübergreifend testen
Für Anwendungen, die global eingesetzt werden, ist es wichtig, länder- und zeitzonenübergreifend Performanz- und Lasttests durchzuführen. Durch Benutzer in unterschiedlichen Zeitzonen können über den ganzen Tag hinweg „Lastwellen" auf die Anwendung „zurollen", die vom System aufgefangen werden müssen.

Spannende Fragen zum Performanz-/Lasttest im mobilen Umfeld werden im Abschn. 15.2.5 behandelt, wie z. B.:

- Wie kommen wir in unterschiedliche Netzwerke?
- Wo bekommen wir die SIM-Karten her?
- Wie viele mobile Geräte müssen wir besorgen?

12.12 Zusammenfassung

- Effizienztests prüfen die Effizienz eines Systems. Dazu gehören Performanztest, Lasttest, Stresstest, Skalierbarkeitstest, Speicherlecktest und Volumentest sowie die Bewertung der Performanzeigenschaften von einzelnen Webseiten und Komponenten.
- Performanz-/Lasttests stellen fest, ob unter normalen Bedingungen Antwortzeiten und Ressourcenverbrauch zufriedenstellend sind und sich unter Last nicht maßgeblich verschlechtern.
- Der Stresstest findet heraus, wann ein System zusammenbricht.
- Der Skalierbarkeitstest stellt sicher, dass steigende (oder fallende) Anforderungen an eine Software allein durch Hardware-Anpassungen erfüllt werden können.
- Performanz-/Lasttests werden in einer produktionsnahen Systemumgebung mit Load Test Tools durchgeführt. Schwachstellen werden mit Monitoren geortet.
- In Performanz-/Lasttestszenarien von mobilen Websites und mobilen Apps müssen die Qualität der Netzverbindungen und der Ladezustand des Akkus berücksichtigt werden.
- Für mobile Anwendungen muss für eine zufriedenstellende Performanz die Anzahl der Request minimiert werden.
- Performanzeigenschaften von Webseiten werden mit statischen Tools anhand von Regelwerken analysiert und bewertet.
- Mit Profilern und Memory-Checkern wird der Zeit- und Speicherplatzverbrauch auf Code-Ebene überprüft.

Tests zur Zuverlässigkeit

13

Das Problem mit der Zuverlässigkeit ist ja vor allem, dass man sich zu schnell an sie gewöhnt.

Peter Rudl (*1966)

Zur Zuverlässigkeit einer Anwendung gehört, dass sie gegenüber fehlerhaften Eingaben tolerant ist. Fehlerhafte Eingaben aus fachlicher Sicht werden beim Testfallentwurf erkannt und mit Testfällen belegt, wie z. B. durch ungültige Äquivalenzklassen und ihre Grenzwerte, s. Kap. 4.1. Diese Negativtestfälle fallen quasi beim Funktionstest ab und werden auch dort ausgeführt, sie werden daher in diesem Kapitel nicht weiter betrachtet – sind aber genau genommen Zuverlässigkeitstests.

Darüber hinaus müssen aus technischer Sicht weitere Tests zur Zuverlässigkeit durchgeführt werden. Der Ausfallsicherheitstest sorgt dafür, dass ein System auch in unvorhergesehenen Situationen beherrschbar bleibt und keine Daten verloren gehen. Der Verfügbarkeitstest misst die Zuverlässigkeit eines Systems während der Betriebszeit. Ausfallsicherheits- und Verfügbarkeitstests beziehen sich auf den Serverbetrieb, d. h. den Back-End-Bereich.

Mobile Endgeräte haben es nicht leicht. Netz- und Funkverbindungen schwanken oder reißen ab. Permanent hat jemand was zu melden (E-Mail, SMS, MMS, Anruf, …). Der Besitzer wechselt Standort, Batterie, SIM-Karte, oder Speicherkarte wann immer er meint, dass es notwendig ist – auch ohne eine App vorher zu beenden, da kennt er nix. Daher ist der Zuverlässigkeitstest im Mobile-Umfeld besonders wichtig.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_13

13.1 Ausfallsicherheitstest

Zur Zuverlässigkeit eines Systems gehört, dass es gegen Ausfall gesichert ist. Sollte das System trotz aller Vorsorgemaßnahmen doch einmal ausfallen, muss gewährleistet sein, dass keine Daten verloren gehen und das System wieder in einem wohldefinierten Zustand gestartet werden kann.

13.1.1 Redundanzen und Failover-Verfahren

Ausfallsicherheit wird durch den Einsatz von Redundanzen erhöht. Dazu werden Systemkomponenten doppelt oder dreifach ausgelegt, damit im Notfall ein Failover stattfinden kann.

Bei einem **Failover** übernimmt eine Backup-Komponente ohne menschliches Eingreifen die Aufgaben einer fehlerhaft arbeitenden oder ausgefallenen Komponente. Ein Failover ist der automatische und ungeplante Wechsel von einer primären Systemkomponente auf eine redundante, sekundäre Systemkomponente.

Bei den betroffenen Systemkomponenten eines Failover kann es sich um die Anwendung, einen Teil der Anwendung, die Datenbank, die Hardware (Webserver, Mail-Server, Load Balancer, …), die Firewall, das Netzwerk, das komplette System oder sogar das gesamte Rechenzentrum handeln. Zum Ausfallsicherheitstest gehört daher auch der Test, dass bei einem Stromausfall die unterbrechungsfreie Stromversorgung gewährleistet ist, d. h. ein Failover durch die Notstromversorgungseinheiten stattfindet.

Zwei Beispiele für die redundante Auslegung von Systemkomponenten sind im Folgenden beschrieben.

Beispiel – Redundante Webserver

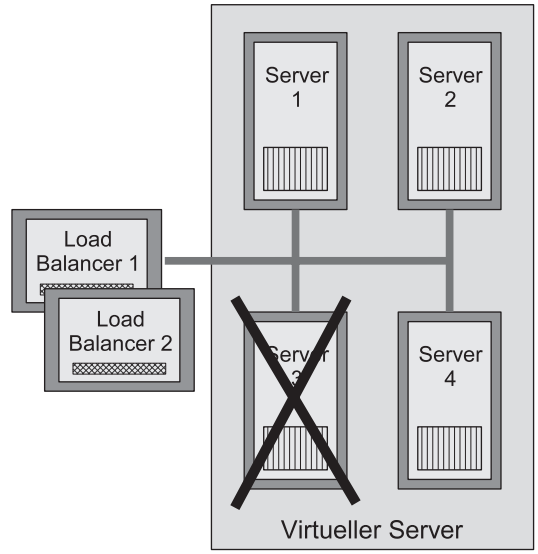
Zwei Server überwachen sich gegenseitig über eine Netzwerkverbindung, indem sie Signale austauschen. Antwortet ein Server nicht, weil er ausgefallen ist, so startet der andere Server automatisch die Übernahme der Dienste.

Beispiel – Redundante Load Balancer

Ein **Load Balancer**[GL] verteilt die anfallende Last auf mehrere Webserver, die zu einem virtuellen Server (Clustered Server[GL]) zusammengeschlossen sind. Fällt einer der Webserver aus, so darf die Anwendung nicht oder zumindest nur unwesentlich davon beeinträchtigt werden.

Testfälle, um die Funktionalität eines Load Balancers zu testen, sind zum Beispiel das Stoppen der Webdienste oder das Ziehen des Steckers eines Servers aus dem Pool des virtuellen Servers (s. Abb. 13.1).

Abb. 13.1 Test Load Balancer



Aber was passiert, wenn der Load-Balancer ausfällt? Auch in diesem Fall muss das System stabil bleiben. Zum Beispiel übernimmt ein zweiter, bereitstehender Load Balancer in einem Failover automatisch die Aufgaben des ausgefallenen. Failover-Verfahren sind Bestandteile der Ausfallsicherheitstests.

► Der **Ausfallsicherheitstest** prüft die Funktionalität und Korrektheit der Failover-Prozesse. Der Ausfallsicherheitstest besteht aus Failover-Test, Failback-Test und Restart-/Recovery-Test.

13.1.2 Failover-Test

► Der **Failover-Test** prüft das rechtzeitige Zuschalten von Ressourcen beim Ausfall einer Systemkomponente und die Funktionsfähigkeit des Systems nach einem Failover.

Für den Failover-Test werden die für den Betrieb der Anwendung kritischen Hard- und Softwarekomponenten bestimmt. Anschließend wird in Testfällen beschrieben, wie der Ausfall der einzelnen Systemkomponenten herbeigeführt wird und wie das System darauf zu reagieren hat.

Zur Testdurchführung werden die automatisierten Testszenarien aus dem Performanz-/Lasttest abgespielt. Zeitgleich wird nacheinander jede der zuvor definierten Fehlersituationen herbeigeführt. Das System wird über einen längeren Zeitraum mit den im Performanztest beschriebenen Werkzeugen beobachtet und ausgewertet.

Eine Verschärfung des Tests besteht darin, Fehlersituationen gleichzeitig hervorzurufen, d. h. den gleichzeitigen Ausfall von mehreren Komponenten herbeizuführen.

13.1.3 Failback-Test

Nach dem provozierten Ausfall einer Systemkomponente (**Failover**) wird die Fehlersituation wieder rückgängig gemacht (**Failback**) und der ursprüngliche Systemzustand wieder hergestellt.

Wie bei einem Failover wird das System nach einem Failback mit den entsprechenden Werkzeugen beobachtet und daraufhin überprüft, ob es wieder die Leistungswerte wie vor dem Failover erreicht.

► Der **Failback-Test** prüft das korrekte Wiederzuschalten einer zuvor ausgefallenen Systemkomponente.

13.1.4 Restart-/Recovery-Test

Wenn trotz aller Sicherheitsvorkehrungen eine Komponente oder gar das gesamte System ausfallen sollte, ohne dass ein Failover stattgefunden hat, muss durch Restart/Recovery-Verfahren gewährleistet sein, dass keine Daten verloren gehen. Datenbanken und Dateisysteme müssen sich nach einem Notfall in einem konsistenten Zustand befinden oder zumindest wieder in einen solchen versetzt werden können.

► **Der Restart-/Recovery-Test (Wiederherstellbarkeitstest)** prüft die Wiederherstellbarkeit eines Systems nach seinem Ausfall. Er stellt die Funktionstüchtigkeit und Korrektheit der festgelegten Restart-/Recovery-Verfahren sicher.

Nach dem Restart des Systems und der Wiederherstellung des ursprünglichen Datenbestandes (Recovery) muss der Benutzer seine Transaktionen mit der Applikation wieder problemlos aufnehmen können. Daten dürfen nicht verloren gegangen, nicht mehrfach angelegt oder fälschlicher Weise verändert worden sein. Falls eingegebene Daten nicht abschließend bearbeitet oder gesichert werden konnten, ist der Benutzer davon zu unterrichten und wieder aufgenommene Funktionen müssen auf einem definierten und konsistenten Datenbestand aufsetzen.

13.1.5 Aktivitäten zum Ausfallsicherheitstest

Der Ausfall von Systemkomponenten muss sowohl auf der Server-Seite als auch auf der Client-Seite getestet werden. Was passiert zum Beispiel, wenn ein Benutzer seinen Client-PC abschaltet oder die Netzverbindung der App während der Kommunikation mit dem Server abbricht? Auch dann dürfen keine Daten verloren gehen und die Datenbestände auf Client und Servern müssen nach einem solchen Ausnahmefall in einem konsistenten Zustand sein. Transaktionen müssen korrekt beendet werden, damit sie nicht von unbefugten

Dritten aufgenommen werden können. Diese Thematik wird auch im Sicherheitstest (s. Kap. 9.10) behandelt.

Die folgende Checkliste fasst die Aktivitäten zum Ausfallsicherheitstest zusammen:

✓ Checkliste Ausfallsicherheitstest

- Wurde eine Ausfallrisikoanalyse durchgeführt?
- Sind darin alle Systemkomponenten berücksichtigt?
- Sind alle kritischen Komponenten doppelt ausgelegt?
- Ist der Ausfall jeder Komponente in mindestens einem Testfall vorgesehen?
- Sind die Failover- und Failback-Zeiten definiert und in die Testerwartungen eingeflossen?
- Ist der Failover-Test erfolgreich durchgeführt und dokumentiert worden?
- Ist der Failback-Test erfolgreich durchgeführt und dokumentiert worden?
- Wurden im Testbetrieb die geforderten Failback-Zeiten nachgewiesen?
- Wurde im Testbetrieb die geforderte Verfügbarkeit[1] nachgewiesen?
- Konnten die vorgegebenen Instandsetzungszeiten im Testbetrieb eingehalten werden?
- Sind Restart/Recovery-Verfahren definiert und geprüft?
- Ist der Restart-/Recovery-Test erfolgreich durchgeführt und dokumentiert worden?

Der Wiederherstellbarkeitstest ist in jedem Fall im Rahmen des Ausfallsicherheitstests durchzuführen, auch wenn keine Failover-Verfahren für das System implementiert sind.

13.1.6 Werkzeuge für den Ausfallsicherheitstest

Der Ausfallsicherheitstest wird mit den Szenarien des Performanz-/Lasttests durchgeführt. Das bedeutet, dass auch hier die im Kap. 12.9 beschriebenen Lasttestwerkzeuge und Monitore zum Einsatz kommen. Zum einen, um die Geschäftsprozesse zu simulieren, zu denen parallel ein Failover oder Failback getestet wird, und zum anderen, um die Messungen und Auswertungen des Systemverhaltens während der Notfallsituationen vorzunehmen.

13.1.7 Qualitätsanforderungen zum Ausfallsicherheitstest

Neben der Erfüllung der Checkliste „Ausfallsicherheitstest“ können konkrete Anforderungen an die Failover- und Instandsetzungszeiten wie in den folgenden Beispielen gestellt werden:

- **Failover-Zeit:** Die Backup-Firewall muss innerhalb von 5 Sekunden die Aufgaben der ausgefallenen Firewall übernehmen.

[1] Verfügbarkeit ist im Abschnitt 11.2.1 definiert.

- **Instandsetzungszeit:** Ist die Anwendung aufgrund eines Systemausfalles nicht verfügbar, muss das System innerhalb von einer Stunde instand gesetzt werden können und die Anwendung wieder zur Verfügung stehen.

13.1.8 Empfehlungen zum Ausfallsicherheitstest

Ausfallrisikoanalyse durchführen
Um ein System ausfallsicher zu machen, können alle Hardwarekomponenten doppelt oder dreifach ausgelegt werden. Bei dieser Lösung ist natürlich die Bedeutsamkeit der Anwendung den entstehenden Kosten gegenüberzustellen. Daher ist es ratsam, eine Risikoanalyse (s. Kap. 6) durchzuführen, in der bewertet wird, welches Risiko der Ausfall einer Systemkomponente bzw. des Systems bedeutet. Daraus werden die entsprechenden Maßnahmen zur Ausfallsicherheit abgeleitet. Ein reines Informationssystem darf sicherlich einige Zeit nicht erreichbar sein, sofern der Benutzer eine entsprechende Meldung erhält. Ein Onlineshop, der plötzlich nicht verfügbar ist, kann fatale Auswirkungen für ein Unternehmen haben.

Ausfallsicherheitstest zum System- und Abnahmetest
Die Messung und die Bewertung der Failover- und Instandsetzungszeiten sollten möglichst früh während des Ausfallsicherheitstests in den Testphasen System- und Abnahmetest begonnen werden, denn so kann rechtzeitig abgeschätzt werden, ob die geforderten Werte in der Produktion theoretisch erreicht werden können. Das ist ein weiteres Argument dafür, Abnahmetests (s. Kap. 16.2) in einer möglichst produktionsidentischen Systemumgebung durchzuführen.

13.2 Verfügbarkeitstest

Zuverlässigkeit (Reliability) ist die Fähigkeit eines Systems, die verlangte Funktionalität unter gegebenen Randbedingungen für eine gegebene Zeit zu erfüllen, also in dieser Zeit ohne Ausfälle zu funktionieren. Um dieses Qualitätsmerkmal messen zu können, wird die Verfügbarkeit definiert.

13.2.1 Definitionen zur Verfügbarkeit

Die DIN 40041 [DIN 40041] beschreibt **Verfügbarkeit** als die Wahrscheinlichkeit, ein System zu einem gegebenen Zeitpunkt in einem funktionsfähigen Zustand anzutreffen. Verfügbarkeit ist demnach das Verhältnis der mittleren fehlerfreien Zeit während des Betrachtungszeitraums zum gesamten Betrachtungszeitraum.

Die mittlere fehlerfreie Zeit während des Betrachtungszeitraums wird als **MTBF (Mean Time Between Failure)** bezeichnet. Dieser Begriff ist sinnvoll, wenn man davon ausgeht, dass ein auftretender Fehler behoben wird und das System nicht für den Rest des Betrachtungszeitraums in dem fehlerhaften Zustand verbleibt.

Die Zeit vom Eintritt des fehlerhaften Zustandes bis zur Wiederherstellung des fehlerfreien Zustandes wird als **MTTR (Mean Time To Repair)** bezeichnet.

Wenn also jeder auftretende Fehler stets behoben wird, ist der Betrachtungszeitraum die Summe aus MTBF und MTTR. **Verfügbarkeit** lässt sich somit darstellen als:

$$Verfügbarkeit = MTBF/(MTBF + MTTR)$$

Als Zahl ausgedrückt, liegt die Verfügbarkeit zwischen 0 und 1. Dementsprechend wird die **Ausfallrate** eines Systems (in der DIN 40041 auch **Unverfügbarkeit** genannt) als Einerkomplement der Verfügbarkeit dargestellt:

$$Ausfallrate = 1- Verfügbarkeit$$

Um eine hohe Verfügbarkeit eines Anwendungssystems zu erhalten, muss die MTTR niedrig gehalten werden, d. h. gegen 0 gehen.

Die MTTR einer Software ist somit auch ein Maß für die **Wartbarkeit** eines Systems, denn nur ein gut wartbares System erlaubt es, Fehler effizient zu lokalisieren und zu beheben.

► Der **Verfügbarkeitstest** prüft die Verfügbarkeit bzw. Ausfallrate eines Systems.

13.2.2 Qualitätsanforderungen und Empfehlungen zum Verfügbarkeitstest

Die Verfügbarkeit des Systems wird in Prozent der Betriebszeit angegeben. Zum Beispiel kann eine 99,5-prozentige Verfügbarkeit vereinbart werden. Von 100 Tagen darf das System also einen halben Tag ausfallen.

Die Messung der Verfügbarkeit ist für den Testbetrieb nur bedingt tauglich, da die tatsächliche Verfügbarkeit eines Systems erst über einen längeren Zeitraum im produktiven Betrieb gemessen werden kann. Aber schon während des Abnahmetests und eines Pilotbetriebes (s. Kap. 16) sollten Verfügbarkeit und Ausfallrate vorgegeben, gemessen und für die Produktion hochgerechnet werden.

13.3 Mobile-Zuverlässigkeitstest

Anwendungen im mobilen Umfeld müssen zuverlässig arbeiten, wenn Störungen oder technische Instabilitäten auftreten – und davon gibt es eine ganz Menge. Einige technische Ausnahmesituationen wurden bereits beim Mobile-Installationstest (s. Abschn. 11.2.2) betrachtet.

Für den Zuverlässigkeitstest einer Mobile-App müssen **Stör(test)fälle** (**Interrupts**) definiert und durchgeführt werden, die von anderen Anwendungen oder vom Benutzer verursacht werden können. Technische Instabilitäten bis hin zu technischen Ausfällen müssen kontrolliert behandelt werden. Eine Applikation bzw. das gesamte System muss auf Unterbrechungen „vernünftig" reagieren.

✓ Checkliste Mobile-Störfälle

Störfälle von anderen Anwendungen

- Eingehende Anrufe, E-Mails, SMS oder MMS
- Terminbenachrichtigung
- Weckeralarm
- Meldung des Batteriezustands

Technische Instabilitäten

- Kurze Unterbrechung der Netzwerkverbindung (Funk und WLAN)[2]
- Ausfall der Netzwerkverbindung
- Funkzellenwechsel beim eigenen Netzanbieter (Roaming[GL])
- Grenzüberschreitendes Roaming mit anderen Anbietern
- Batterieausfall
- Automatische Zeitabschaltung des Gerätes
- Voller Speicher des Endgerätes

Benutzeraktionen im laufenden App-Betrieb

- SIM-Karte wechseln
- Speicherkarte wechseln
- Gerät mit Desktop-PC via USB-Kabel verbinden
- USB-Kabel zum Desktop-PC trennen
- App unterbrechen (z. B. via Home-Taste des Gerätes)
- Online- auf Offline-Betrieb umschalten und zurück
- Downloads und Uploads abbrechen und erneut starten
- Sonstige Aktionen abbrechen.

[2] Für diesen Test empfehle ich die B43 am Flughafen in Richtung Frankfurt, kurz hinter „The Squaire". Hier bricht im Auto die Netzverbindung meines Providers ab, immer.

Für jeden Störfall müssen Testfälle mit dem erwarteten Verhalten entworfen und durchgeführt werden.

Als erwartete Reaktionen kann die zu testende App zum Beispiel Daten speichern, gesteuert beendet oder in einen Pause-Modus versetzt werden. Eine „geordnete Rückkehr“ zur verlassenen App muss je nach Störung und Anforderung automatisch erfolgen oder manuell möglich sein. Der Benutzer muss über die aktuelle Situation informiert werden.

13.4 Zusammenfassung

- Ausfallsicherheitstest und Verfügbarkeitstest prüfen die Zuverlässigkeit des Gesamtsystems.
- Der Ausfallsicherheitstest testet die Zuverlässigkeit eines Systems und besteht aus Failover-, Failback- und Restart-/Recovery-Test.
- Der Restart-/Recovery-Test stellt sicher, dass die Datenbasis auch nach Systemstörungen in einem konsistenten Zustand ist.
- Redundante Systemkomponenten und Failover- und Failback-Maßnahmen erhöhen die Ausfallsicherheit eines Systems.
- Verfügbarkeit ist der Grad, zu dem eine Komponente oder ein System für die Nutzung zur Verfügung gestanden hat. Sie ist ein Maß der Zuverlässigkeit und wird im Verfügbarkeitstest überprüft.
- Der Mobile-Zuverlässigkeitstest prüft die Zuverlässigkeit von Apps auf und mit dem (mobilen) Endgerät und im Zusammenspiel mit dem Netz.

Teil III
Testplanung

14. Testwiederholungen
15. Planung der Testarten
16. Planung der Teststufen
17. Planung der Testteams

Im dritten Teil des Buches werden die Testprozesse aus Sicht des Testmanagements beschrieben, das die notwendigen Maßnahmen zur Qualitätssicherung von Software-Anwendungen planen und steuern muss.

Nach Änderungen in der Software oder am Systemumfeld müssen Tests wiederholt werden. Die bei der Testplanung zu berücksichtigenden Möglichkeiten und Aspekte sind Thema des Kap. 14, Testwiederholungen.

Im Testkonzept eines Projektes wird festgelegt, welche Testarten mit welcher Intensität durchgeführt und welche Testmittel dafür eingesetzt werden sollen. Die Kriterien für die zu treffenden Entscheidungen werden im Kap. 15, Planung der Testarten, erläutert.

Im Kap. 16 werden die zeitlichen Abschnitte beschrieben, die in einem Software-Entwicklungsprojekt geplant und durchlaufen werden, sei es sequentiell oder in Iterationen.

Last but not least, die Testarbeit muss von jemandem getan werden. Welche Personen mit welchem Wissen und welchen Erfahrungen in einem Testteam eingeplant werden müssen, wird im Kap. 17, Planung des Testteams, beschrieben.

Testwiederholungen

14

Die Reparatur alter Fehler kostet oft mehrals die Anschaffung neuer.

Wieslaw Brudzinski (*1920)

Wiederholungen von bereits durchgeführten Tests sind notwendige Tätigkeiten, deren Aufwand bei der Testplanung nicht unterschätzt werden darf.

Wiederholungstests werden in Fehlernachtest und Regressionstest unterschieden. Sie sind keine Testarten im Sinne der im Teil II des Buches beschriebenen Tests, denn sie stellen „nur“ indirekt die Qualitätsmerkmale einer Software sicher.

14.1 Fehlernachtest

► Mit einem **Fehlernachtest** wird geprüft, ob ein Fehlerzustand korrekt beseitigt wurde.

Beim Fehlernachtest werden nur diejenigen Testfälle wiederholt, die eine Fehlerwirkung aufgedeckt haben und deren Fehlerzustand inzwischen behoben wurde.

Ob einem Fehlernachtest ein umfassenderer Regressionstest folgen muss, ist von der Bedeutung des behobenen Fehlers abhängig und muss von Fall zu Fall vom Testmanagement im Rahmen der Teststeuerung entschieden werden.

Wurde zum Beispiel auf der Benutzeroberfläche eine Zahl mit einer statt mit zwei Nachkommastellen dargestellt, genügt ein lokaler Fehlernachtest. Wurde die Zahl dagegen aufgrund einer fehlerhaft programmierten Formel falsch berechnet, dann ist, wenn die Formel in anderen Programmteilen der Anwendung benutzt wird, ein entsprechender Regressionstest notwendig.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_14

14.2 Regressionstest

► Der **Regressionstest** ist die Wiederholung der Tests eines bereits getesteten Testobjekts (Komponente, App, System, …) nach einer Modifikation. Er soll sicherzustellen, dass durch die vorgenommenen Änderungen keine neuen Fehler entstanden sind und dass bisher maskierte Fehlerwirkungen aufgedeckt werden.

Für den Regressionstest werden keine besonderen Testfälle entwickelt, sondern bestehende Testfälle, die bei der letzten Durchführung keine Abweichungen geliefert haben, werden wiederholt. Der Regressionstest untersucht die Abweichungen der Testergebnisse im Vergleich zum Testlauf mit der Vorgängerversion des Testobjektes.

Je nachdem, welche Systemkomponente in welcher Projektphase modifiziert wurde, kann je nach Software-Entwicklungsmodell ein Regressionstest in jeder Testphase (s. Kap. 16) notwendig sein. Drei Ereignisse können einen Regressionstest erforderlich machen:

1. Bekannte Fehler wurden behoben.
2. Neue Funktionen wurden realisiert oder bestehende wurden geändert.
3. Die Systemlandschaft wurde verändert.

Regressionstests nach Fehlerbehebung
Bei der Korrektur von Fehlerzuständen kann es vorkommen, dass neue entstehen. Zudem können vorhandene Fehlerzustände bisher nicht gefunden worden sein, weil die Fehlerwirkungen von anderen maskiert worden sind. In einem Regressionstest wird geprüft, ob die Anwendung nicht „verschlimmbessert“ wurde und ob bisher maskierte Fehlerwirkungen auftreten.

Die Testwiederholung aufgrund eines behobenen Fehlers beschränkt sich nicht nur auf die Komponente, in welcher der Fehler korrigiert wurde; zumindest sind die mit der modifizierten Komponente direkt kommunizierenden Systemkomponenten zu überprüfen.

Soll nur ein begrenzter Regressionstest durchgeführt werden, muss das Testmanagement das Risiko gegen den Aufwand eines kompletten Regressionstests abwägen und eine begründete Entscheidung treffen, warum ein kompletter Regressionstest nicht notwendig ist.

Regressionstests nach Funktionsveränderungen
Wenn eine Anwendung um neue Funktionen erweitert oder wenn bestehende Funktionen verändert werden, müssen diese neuen bzw. geänderten Funktionalitäten nach den in den vorhergehenden Kapiteln beschriebenen Testverfahren einzeln getestet werden.

Anschließend muss ein umfangreicher Regressionstest sicherstellen, dass keine der bestehenden, unveränderten Funktionen durch die Neuerungen ungewollt beeinflusst werden. Der Regressionstest nach funktionalen Änderungen erstreckt sich i. d. R. über alle bisher durchgeführten Testphasen, bis hin zum erneuten Abnahmetest (s. Kap. 16).

In agilen und testgetriebenen Entwicklungsprojekten sind zumindest auf komponentenebene per Definition Regressionstests für jeden Entwicklungsschritt vorgesehen und schon vom Entwickler automatisiert auszuführen (s. Kap. 9.1).

Regressionstests nach Systemveränderungen
Wenn sich die Systemkonfiguration oder das technische Umfeld, in der eine Anwendung betrieben wird, ändert, ist für bestimmte Bereiche des Systems eine Testwiederholung fällig. Dazu einige Beispiele:

- Wird die Anwendung auf eine neue Datenbank umgestellt, müssen Tests für die Komponenten wiederholt werden, die auf die Datenbank zugreifen. Performanz-, Last- und Restart-/Recovery-Tests müssen erneut durchgeführt werden.
- Wird eine neue Hardware-Generation der Server installiert, müssen alle Performanz-/Lasttests wiederholt werden.
- Wird die bestehende Client-Server-Kommunikation auf das neue Konzept AJaX[GL] umgestellt, müssen alle Funktionstests überarbeitet und die Usability neu bewertet werden. Denn mit AJaX wird die Benutzerfreundlichkeit verbessert, weil der Benutzer Aktionen auf der Website ausführen kann, während die Daten vom Server asynchron nachgeladen werden.
- Kommt ein neues mobiles Endgerät auf den Markt, so muss der Interoperabilitätstest wiederholt werden.
- Soll die Anwendung unter einem neuen Betriebssystem eingesetzt werden, sind zumindest die Tests zu wiederholen, die sich auf den Client beziehen (Cross-Browser-Test, Plugin-Test, Sicherheitstest und ggf. Installationstest).
- Der Relaunch einer Website zieht einen kompletten Regressionstest nach sich.

Beispiel: Regression des Plugin-Tests

Die Notwendigkeit des Regressionstests wird an dem Beispiel in Kap. 9.7, **„Eine wahre Plugin-Testgeschichte**“, deutlich. Wir erinnern uns, dass die Web-App ein Java-Plugin ab einer bestimmten Version benötigt und dass das Plugin inzwischen erfolgreich installiert und ohne Probleme im Einsatz ist. Die Testgeschichte nimmt nun ihren Lauf.

Szenario Plugin-Test – Teil 5

Ein neues Betriebssystem für Clients kommt auf den Markt. Der Regressionstest der Anwendung wird mit dem neuen Betriebssystem durchgeführt. Das Ergebnis zeigt, dass unser Java-Applet mit dem Java-Plugin 1.4.1 von diesem neuen Betriebssystem nicht geladen wird.

Warum? Die Kompatibilität ist seitens Hersteller nicht gewährleistet. Erst ein Service Pack zum Plugin schafft Abhilfe. Dem Benutzer wird umgehend diese Information zur Verfügung gestellt und noch besser: Stellt die Anwendung beim Start fest, dass ein

Benutzer das neue Betriebssystem ohne das notwendige Service Pack installiert hat, wird dieses automatisch aus dem Internet heruntergeladen und installiert. Das ist natürlich eine neue Anforderung an die Webanwendung, die spezifiziert, programmiert und getestet wird.

Szenario Plugin-Test – Teil 6

Der Fall geht noch weiter: Monate später gibt der Hersteller eine neue Version 2.0 des Plugins frei, die zu allen Vorgängerversionen des Plugins abwärtskompatibel sein soll.

Ha! Der Regressionstest zeigt, dass Funktionen, die Swing-Klassen[GL] benutzen, nicht mehr korrekt arbeiten. Das Java-Applet der Webanwendung funktioniert nicht mehr richtig. Und die alte Version des Plugins wird vom Hersteller nicht mehr unterstützt.

14.3 Werkzeuge für die Testwiederholung

Es liegt nahe, dass Tests nur effektiv und bezahlbar in größerem Umfang und mehrmals wiederholt werden können, wenn sie automatisiert sind. Zur automatisierten Testwiederholung werden Unit-Test Tools, Load Test Tools und Capture Replay Tools eingesetzt.

Testframeworks für Unit-Tests sind im Abschn. 9.1.4 genannt. Load Test Tools (Lasttestwerkzeuge), mit denen Regressionstests zur Performanz und Ausfallsicherheit automatisiert werden, sind ausführlich im Kap. 12.5 erläutert. Daher werden an dieser Stelle nur noch die Testroboter beschrieben.

Tests, die über die Benutzeroberfläche der Anwendung durchgeführt werden, können mit Capture Replay Tools – auch Testroboter genannt – automatisiert werden.

Capture Replay Tools erkennen die Objekte auf der Benutzeroberfläche der zu testenden Anwendung (Texte, Buttons, Eingabefelder, Grafiken, …) und zeichnen bei der Testdurchführung die Eingaben, Mausklicks und Angaben des Benutzers auf. Der Tester legt bei der Aufzeichnung fest, welche Objekte, Eigenschaften der Objekte und Daten später bei der Wiederholung des Tests auf Veränderungen überprüft werden sollen.

Beim Regressionstest werden die aufgezeichneten und ggf. durch Programmierung erweiterten Testskripte automatisch vom Testwerkzeug abgespielt. Abweichungen gegenüber den vorangegangenen Aufzeichnungen werden protokolliert. Der Tester muss dann entscheiden, ob eine ausgewiesene Veränderung gewollt war oder nicht.

Die einzelnen Regressionstest-Tools haben unterschiedliche Fähigkeiten und Einsatzmöglichkeiten. Komfortable Werkzeuge erlauben es, mehrere Browser-Typen und unterschiedliche grafische Oberflächen mit denselben Testskripten zu testen. Testskripte können so programmiert werden, dass die Tests nicht komplett aufgezeichnet werden müssen, sondern parallel zur Programmierung erstellt werden können. Zudem können Datei- und Datenbankinhalte automatisiert verglichen werden.

Bekannte kommerzielle Schwergewichte unter den Testautomatisierungswerkzeugen sind **HP QuickTest Professional** von Hewlett-Packard oder **Rational Functional Tester** von IBM, schwergewichtig im Umfang und im Verbreitungsgrad.

Abhängig von den Anforderungen, die an eine Webanwendung gestellt werden, können kostengünstigere (wie z. B. QF-Test) oder gar kostenlose (wie z. B. Selenium) Capture Replay Tools eine Alternative zu kommerziellen Testrobotern sein.

QF-Test von der Quality First Software GmbH ist auf Java- und Webanwendungen spezialisiert [URL: QF-Test]. Eine interessante Beschreibung für datengetriebene und schlüsselwortgetriebene Testautomatisierung anhand eines Praxisbeispiels mit QF-Test hat Dirk O. Schweier verfasst, [Schweier_QF-Test_2013].

Beim **datengetriebenen Testen** werden die Testeingaben und vorausgesagten Ergebnisse in einer Datei (z. B. Excel-Tabelle, XML- oder CSV-Datei) gespeichert. Der Testroboter führt ein bestimmtes Testskript mit jedem Datensatz aus der Datei aus. Man kann das Testskript als abstrakten Testfall verstehen, der zusammen mit einem Datensatz zu einem konkreten Testfall wird. Die Daten treiben den Test an.

Beim **schlüsselwortgetriebenen Testen** enthalten diese testtreibenden Dateien zusätzlich Schlüsselworte. Das Testskript interpretiert diese Schlüsselworte und kann damit während des Testlaufs unterschiedliche Testmodule mit den dazugehörenden Testdaten ansteuern. Die Schlüsselworte steuern den Test.

Auf die Testautomatisierung von .NET-Anwendungen und unter den Betriebssystemen Android und iOS laufenden Apps hat zum Beispiel **Ranorex** der Ranorex GmbH Schwerpunkte gelegt, [URL: Ranorex].

Ein Open Source Tool unter Apache License zur Automatisierung von Funktions- und Regressionstests ist **Selenium** [URL: Selenium]. Selenium besteht aus mehreren Werkzeugen. Eines davon ist Selenium-IDE, das eine komplette Testumgebung als Firefox-Plugin bereitstellt.

Abbildung 14.1 zeigt ein Beispielskript von Selenium-IDE. Darin sind im Reiter „Table“ das aufgezeichnete Skript zu sehen und darunter der im Skript selektierte Befehl, der an dieser Stelle editiert werden kann. Im unteren Teil des Fensters wird das Protokoll der letzten, automatisch ausgeführten Testwiederholung aufgeführt.

Umfassende Übersichten von Capture Replay Tools sind zum Beispiel unter [URL: TestToolsOPENSOURCE] Rubrik „Testing tools – Functional testing“ oder [URL: TestToolsSQA] Rubrik „Web Tools – Web Functional/Regression Test Tools“ zu finden. [URL: TestToolsSQA] listet unter „Mobile Web/App Testing Tools“ Tools zur Testautomatisierung von mobilen Anwendungen auf.

14.4 Empfehlungen zur Testwiederholung

Testwiederholungen planen

Will man nach einer Änderung auf der sicheren Seite sein, müssten alle bisher durchgeführten Tests wiederholt werden. Das ist natürlich nicht immer notwendig und auch nicht bezahlbar. Daher müssen für jede Veränderung einer Software die Testwiederholungen bewertet und neu geplant werden.

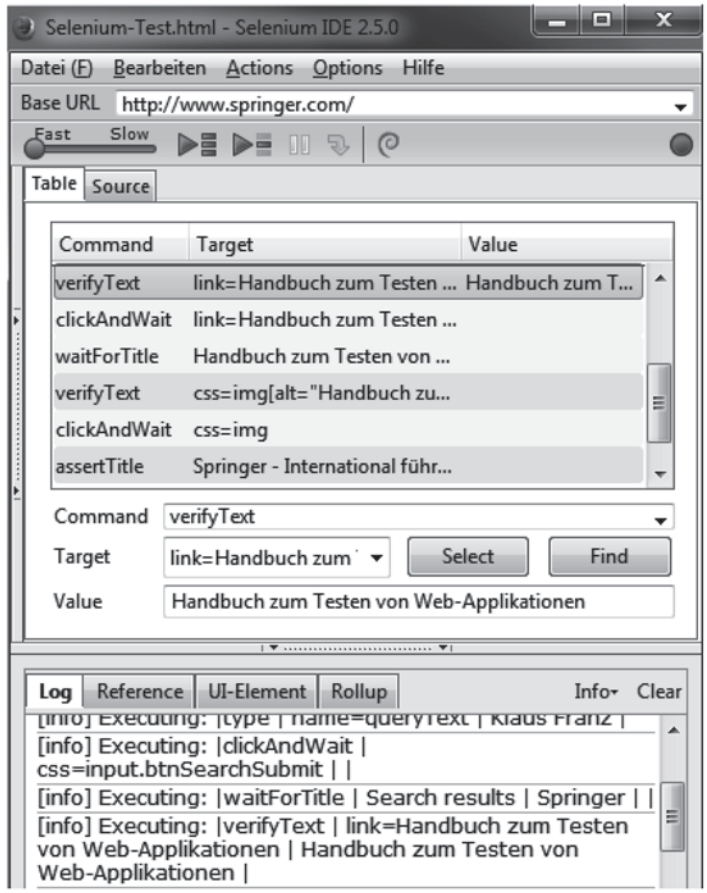


Abb. 14.1 Selenium

Unit-Tests mit Tools
Wird eine Anwendung testgetrieben entwickelt, muss mit jeder Sourcecode-Ergänzung der bisherige Code einem Regressionstest unterzogen werden. Die Entwicklung führt bei diesem Vorgehen auch schon permanent Testwiederholungen mit Unit-Test Tools durch, s. Kap. 9.1.

Capture Replay für Ersttests
Capture Replay Tools ermöglichen schon in frühen Testphasen eine Automatisierung von Tests, indem die entworfenen Testfälle in programmierten Testskripts abgebildet und ausgeführt werden. So können nicht nur die später notwendigen Regressionstests, sondern auch die ersten Testläufe für eine Komponente automatisch ablaufen.

Testautomatisierungsspezialisten
In Projekten hat sich gezeigt, dass Testroboter zur Testautomatisierung nicht von jedem Tester im Testteam eingesetzt werden sollten, weil spezielle Programmierkenntnisse zum optimalen Einsatz dieser Tools notwendig sind. Auch wenn der Trend zum benutzerfreundlichen – oder besser gesagt zum testerfreundlichen – Aufzeichnen und Abspielen der Testszenarien geht, ist es i. d. R. effektiver, wenn ein Spezialistenteam für die Testautomatisierung zuständig ist. Dieses Team übernimmt die Erstellung und Pflege aller Testskripte. Der Aufwand für die Anpassung von Testskripten durch Programmänderungen oder neue Versionen des eingesetzten Testtools ist weder von der Komplexität noch vom Umfang her zu unterschätzen.

Nicht alles automatisieren
Nicht alle Tests müssen unbedingt automatisiert werden. Weil die Ersterstellung und die Pflege von Testskripten aufwändig ist, muss bei der Testplanung genau überlegt werden, für welche Testszenarien sich ein automatisierter Regressionstest rechnet. Kandidaten für automatisierte Tests sind natürlich kritische Funktionen, die bei jeder Programmanpassung überprüft werden müssen. Dass sich Testautomatisierung auszahlen kann, zeigt das folgende Beispiel.

Wie sich Testautomatisierung auszahlen kann

Ein Programm – es ist hoffentlich noch immer im Einsatz – berechnet anhand vorgegebener Formeln die korrekten Parametereinstellungen für technische Messgeräte. Aufgrund der gültigen Äquivalenzklassen und deren möglichen Kombinationen ergeben sich ca. 400 Berechnungsmöglichkeiten.

Für die Durchführung der Tests war eine Reduzierung der zu testenden Kombinationen zu riskant, weil jede Kombination ein anderes Ergebnis liefert. Um die 400 Testszenarien effektiv wiederholen zu können, wurden die Tests automatisiert.

Folgendes Fazit kann gezogen werden: Die Testautomatisierung brachte erst einmal keine Zeitersparnis gegenüber einem manuellen Test, denn die im ersten Testlauf eingesparte Zeit war zuvor für die Programmierung und die Tests der Testskripte verbraucht worden.

Aber: Unter optimalen Bedingungen würde ein manueller Regressionstest über alle 400 Testfälle durch eine erfahrene Person vier Tage dauern. Ein automatisierter Testdurchlauf benötigt zwei Stunden!

Und allein im Laufe des Projektes wurde der Regressionstest fünfmal durchgeführt. Hinzu kamen (und kommen immer noch) die Testläufe im Rahmen von Wartungsarbeiten. Das hat sich gelohnt!

14.5 Zusammenfassung

- Testwiederholungen sind notwendig, nachdem bekannte Fehler behoben, neue Funktionen realisiert, bestehende Funktionen geändert oder die Systemlandschaft verändert wurden.
- Der Fehlernachtest ist die Wiederholung eines Testfalls, der einen Fehler aufgedeckt hat.
- Der Regressionstest ist die umfassende Testwiederholung bereits durchgeführter Tests. Er stellt in der Testumgebung sicher, dass sich keine neuen Fehler in das System eingeschlichen haben.
- Bei der Testplanung muss festgelegt werden, welche Regressionstests automatisiert werden müssen und welche effizienter manuell durchgeführt werden können.
- Regressionstests, die sich auf die Oberfläche einer Webanwendung beziehen, werden mit Capture Replay Tools automatisiert. Dabei werden aufgezeichnete und durch Programmierung erweiterte Testskripte wiederholt abgespielt.
- Beim Unit-Test werden schon bei der Entwicklung permanent Testwiederholungen mit Unit-Test Tools durchgeführt.
- Für die automatische Wiederholung von Performanz-/Lasttests werden Load Test Tools eingesetzt.
- Für die Automatisierung von Regressionstests werden Testtool-Spezialisten benötigt.

Planung der Testarten

15

Jede Dummheit findet einen, der sie macht.

Tennessee Williams (1911–1983)

Um die Tests zu einer Anwendung detailliert planen zu können, muss geprüft werden, welche Testarten für das Projekt relevant sind. Daraus leitet sich ab, mit welcher Intensität, welchen Verfahren und mit welchen Testmitteln die Tests durchgeführt werden sollen.

15.1 Bewertung der Testarten

Im Rahmen der Testplanung wird festgelegt, welche Testarten mit welcher Intensität eingesetzt werden sollen. Je nachdem, welche Ziele verfolgt und welche Qualitätsanforderungen an die Anwendung gestellt werden und in welchem technischem Umfeld diese betrieben werden soll, sind die Tests zu planen und durchzuführen. Ein paar Beispiele:

- Für eine Website, die hauptsächlich Informationen bereitstellt, ist ein Ausfallsicherheitstest nicht von großer Bedeutung, dafür aber ein Auffindbarkeitstest.
- Wird eine Applikation in einem Intranet betrieben, in dem alle Benutzer den gleichen Browser benutzen, muss kein Cross-Browser-Test über alle existierenden Browser-Typen stattfinden. Wenn der Nutzerkreis aus der weltweiten Webgemeinde besteht, haben Cross-Browser- und Zugänglichkeitstests eine hohe Priorität.
- Web-Apps für Behörden erfordern intensive Usability- und Zugänglichkeitstests.
- Mobile-Web-Apps, die auf (fast) beliebig vielen mobilen Endgeräten laufen sollen, müssen einem intensiven Interoperabilitätstest unterliegen.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_15

- Soll die Anwendung bei mehreren Kunden eingesetzt werden und muss sie deshalb individuell angepasst werden können, sind Code-Inspektionen, Code-Walkthroughs und Installationstests wichtig, um ihre Änderbarkeit und Übertragbarkeit zu gewährleisten.
- Für Banking-Systeme sind Sicherheit, Ausfallsicherheit und Verfügbarkeit essentielle Qualitätsanforderungen.

Die Liste der Beispiele lässt sich beliebig fortführen…

In Tabelle 1.1 sind alle in diesem Buch beschriebenen Testarten aufgeführt. Sie dient als Grundlage für die Planung der in einem Projekt durchzuführenden Tests.

Bewertung von Testarten für den Finanzierungsrechner

In Tab. 15.1 ist für unser Beispiel aus den vorangehenden Kapiteln, dem Finanzierungsrechner, exemplarisch eine Bewertung aller Testarten vorgenommen worden. Der Finanzierungsrechner ist als Web-App für die Nutzung auf Desktop-Rechnern entworfen.

Wenn Alternativen in der Durchführung einer Testart bestehen, muss das Testmanagement in Abhängigkeit der vorgenommenen Bewertung bestimmen, welches konkrete Verfahren eingesetzt werden soll. Unter anderem müssen folgende Fragen vor der Erstellung des Testkonzeptes beantwortet werden:

Dokumententest
Für welche Typen von Dokumenten müssen nur formale Prüfungen durchgeführt werden und für welche zusätzliche Review-Sitzungen oder schriftliche Stellungnahmen?

Unit-Test/Komponententest
Soll das Entwicklungsteam im Unit-Test Testfälle automatisierten und wenn ja, welche? Oder liegt der Komponententest vollständig beim Testteam?

Sollen Testüberdeckungsgrade nachgewiesen werden? Wenn ja, für welche Testüberdeckung (Methodenüberdeckung, Anweisungsüberdeckung, Zweigüberdeckung, eine der Bedingungsüberdeckungen)?

Integrationstest
Nach welchen Verfahren werden Komponenten und Subsysteme integriert (Ad-hoc, Anwendungsfallbasiert, Backbone, Continuous Integration)?

Interoperabilitätstest (Cross-Browser-, Cross-Device-Test)
Soll die Website/Web-App für mobile Geräte geeignet sein? Wenn ja, welche Browser, Betriebssysteme und Endgeräte sollen unterstützt werden? Auf welchen Endgeräten soll die Mobile-App installiert werden können?

Tab. 15.1 Beispielbewertung der Testarten für den Finanzierungsrechner

Testart	Bewertung	Begründung
Dokumententest	Hoch	Anforderungsspezifikationen enthalten geschäftsrelevante Funktionsbeschreibungen
Tests zur Funktionalität		
Unit-Test	Mittel	Unit-Test durch das Entwicklerteam inkl. Code-Analysen, es findet ein intensiver Komponententest statt
Funktionaler Komponententest	Hoch	Geschäftsrelevante Funktionen, daher intensiver Komponententest durch das Testteam
Integrationstest	Hoch	Integration geschäftsrelevanter Subsysteme
Funktionaler Systemtest	Hoch	Geschäftsrelevante Funktionen
Link-Test	Niedrig	Keine externen Links vorgesehen
Cookie-Test	Hoch	Geschäftsrelevante Funktionen setzen Cookies ein
Plugin-Test	Hoch	RIA-Technologien werden eingesetzt
Test der Browser-Einstellungen	Hoch	Wichtig für die Funktionalitäten
Webservice-Test	Niedrig	Die Währungsumrechnung via Webservice ist nicht geschäftsrelevant
Sicherheitstest	Hoch	Sensible Daten werden verarbeitet
Cross-Browser-Test	Hoch	Alle gängigen Desktop-Browser sollen unterstützt werden
Cross-Device-Test	Niedrig	Finanzierungsrechner ist für Nutzung auf Desktop-Rechnern vorgesehen
Mobile-Funktionstest	Mittel	Nur Mobiltauglichkeit ist gefordert, keine Mobilfähigkeit
Tests zur Benutzbarkeit		
Content-Test	Hoch	Geschäftsrelevanter Web-Auftritt
Oberflächentest	Hoch	Unternehmensstandards sind einzuhalten
Usability-Test	Hoch	Kundenzufriedenheit ist ein kritischer Erfolgsfaktor
Zugänglichkeitstest	Mittel	Zugänglichkeit ist nicht explizit gefordert
Auffindbarkeitstest	Niedrig	Benutzer kommen über das gut besuchte Firmenportal
Tests zur Änderbarkeit und Übertragbarkeit		
Code-Analysen	Mittel	Finden im Rahmen des Unit-Tests statt
Installationstest	Nicht relevant	Anwendung wird im Hause betrieben, keine Client-Installationen notwendig
Mobile-Installationstest	Nicht relevant	–
Tests zur Effizienz		
Performanztest	Hoch	Verarbeitung von geschäftsrelevanten Transaktionen, Performanz ist kritischer Erfolgsfaktor (ca. 2000 User)

Tab. 15.1 (Fortsetzung)

Testart	Bewertung	Begründung
Last- und Stresstest	Hoch	Zugriffe werden durch Marketingaktionen erfahrungsgemäß für ca. eine Woche verdreifacht (Stresstest für ca. 8.000 User)
Skalierbarkeitstest	Niedrig	Keine Aufrüstung geplant
Speicherlecktest	Niedrig	Speicherverwendung wird im Unit-Test geprüft
Mobile-Performanz-/Lasttest	Nicht relevant	–
Tests zur Zuverlässigkeit		
Ausfallsicherheitstest	Niedrig	Sicherheitsvorkehrungen zum bestehenden Firmenportal sind etabliert und geprüft
Verfügbarkeitstest	Hoch	24/7 Betrieb mit 99 % Verfügbarkeit gefordert

Usability-Test
In welchem Umfang soll die Gebrauchstauglichkeit der Anwendung getestet werden (Usability-Labor inkl. Blickregistrierung und/oder Befragung, Online-Umfrage)?

Zugänglichkeitstest
Soll die Zugänglichkeit nach den Anforderungen der BITV geprüft werden? Wenn ja, genügt der Nachweis der Punkte der Priorität 1?

Code-Analysen
Für welche Komponenten sollen welche Code-Analysen durchgeführt werden (Statische Code-Analyse, Code-Walkthrough, Code-Inspektion, Schreibtischtest)? Wenn ja, durch das Entwicklungsteam?

Ausfallsicherheitstest
Ist zur Überprüfung der Ausfallsicherheit ein anwendungsspezifischer Restart-/Recovery-Test hinreichend, weil die Failover-Verfahren durch das bestehende, produktive Systemumfeld gewährleistet sind? Oder werden neue Failover-Maßnahmen implementiert, die getestet werden müssen?

15.2 Bereitstellung der Testmittel

Nachdem die Testarten in ihren Ausprägungen feststehen, müssen Checklisten, Systemumgebungen, Testwerkzeuge und ggf. mobile Endgeräte für die Testdurchführung ausgewählt und bereitgestellt werden.

15.2.1 Bereitstellung von Checklisten

Zur Planung der QS-Maßnahmen in einem Projekt gehört die Festlegung, welche Checklisten eingesetzt werden sollen. Wenn zum Beispiel die Anwendung in der Programmiersprache PHP (Private Home Page) entwickelt wird und Code-Inspektionen für kritische Programme durchgeführt werden sollen, muss dafür eine entsprechende Checkliste „PHP-Code-Inspektion“ bereit gestellt werden.

Der Testmanager muss prüfen, ob neue Checklisten erstellt werden oder ob existierende Checklisten „nur“ überarbeitet werden müssen. Der Aufwand und die Ressourcen zur Bereitstellung der Checklisten müssen in die Testplanung einfließen.

In der Tab. 1.1 *des Kap. 1 sind die in diesem Buch vorgestellten Checklisten aufgeführt. Sie sollen als fundierte Grundlage für die Erarbeitung projektspezifischer Checklisten dienen. Zum Einsatz von Checklisten als konstruktive und analytische QS-Maßnahme siehe Kap. 6.*

15.2.2 Bereitstellung der Testumgebungen

Je nach Anwendung, Qualitätsanforderungen und technischen Möglichkeiten werden unterschiedliche Testumgebungen benötigt. Diese müssen geplant und bereitgestellt werden.

Die Entwickler müssen in ihrer **Entwicklungsumgebung** testen, je nach Entwicklungsmodell mehr oder weniger (s. Kap. 16).

Für die sich anschließenden Tests durch Testteam(s) und Auftraggeber werden mehrere Testumgebungen benötigt. Welche das sind und ob einige Umgebungen zusammengefasst werden können (oder aus technischen oder organisatorischen Gründen zusammengefasst werden müssen) entscheidet das Testmanagement projektabhängig.

Eine von der Entwicklung getrennte **Testumgebung** wird **für Komponententests** und/oder **Komponentenintegrationstests** benötigt, wenn sie durch ein Testteam ausgeführt werden. Hier werden dann hauptsächlich Tests zur Funktionalität durchgeführt.

Oft werden mehrere **Systemtestumgebungen** benötigt, damit sich die Tests nicht gegenseitig behindern. Je nach Testziel und Personenkreis kann eine spezielle Testumgebung notwendig sein, zum Beispiel:

- für manuelle Tests, in der vom Fachbereich Geschäftsprozesse mit eigenen Testdaten getestet werden,
- für die Testautomatisierung, in der die Testdaten regelmäßig zurückgesetzt werden müssen,
- für Effizienztests in einer produktionsnahen Umgebung, um realitätsnahe Messungen durchführen zu können,
- für Zuverlässigkeitstests, damit das Rechenzentrum z. B. für Restart-Recovery-Tests den Stecker ziehen kann,

- für externe Usability-Tests, damit externe Tester und/oder ein beauftragtes Usability-Labor die Anwendung prüfen kann,
- für den Crowdtest, damit die Test-Crowd (s. Kap. 17.2) im Interoperabilitätstest nichts kaputt machen kann.

Wenn mehrere Teilsysteme zu einem Gesamtsystem integriert werden müssen, wird eine **Systemintegrationstestumgebung** gebraucht.

Je nachdem, wie die Abnahme durch den Auftraggeber vereinbart worden ist, ist eine „schöne", separate **Abnahmetestumgebung** bereitzustellen.

Das Testmanagement muss entscheiden, welche Testumgebungen wann und wem mit welchen Daten zur Verfügung gestellt werden müssen. Dabei spielt neben den Testzielen auch das eingesetzte Software-Entwicklungsmodell eine entscheidende Rolle (s. Kap. 16).

…Und alle benötigten Testumgebungen müssen rechtzeitig budgetiert und beauftragt werden.

15.2.3 Bereitstellung der Testdaten

Eine gern unterschätze Aktivität ist die Planung und Bereitstellung der Testdaten für die einzelnen Testumgebungen. Folgende Fragen müssen im Rahmen des Testdatenmanagements beantwortet werden:

- Für welche Tests können/sollen synthetische Testdaten automatisch erzeugt werden?
- In welchen Testumgebungen müssen anonymisierte Produktionsdatenbestände bereitgestellt werden?
- Wer stellt welche Testdatenbestände wann, wie und wem zur Verfügung?
- Wer gibt welche Testdaten für welchen Test frei (Rechenzentrum, Datenschutzbeauftragter, Fachbereich,…)?
- Mit welchen Werkzeugen werden Testdaten generiert oder anonymisiert?
- Wie werden Testdatenbestände archiviert und für Wiederholungstests zurückgesetzt?

Testdatenmanagement ist ein eigenes, spannendes Thema, das hier nicht weiter vertieft wird.

15.2.4 Bereitstellung der Testwerkzeuge

In Abhängigkeit der geplanten Tests müssen Testwerkzeuge ausgewählt und zur Verfügung gestellt werden. Bei der Planung des Einsatzes von Testtools wird geprüft, ob die Werkzeuge schon im Unternehmen im Einsatz sind oder ob sie erst eingeführt und geschult werden müssen. Ggf. müssen entsprechende Aktivitäten geplant und durchgeführt werden. Eine Vorgehensweise zur Auswahl und Einführung von Testwerkzeugen ist z. B. in [Spillner_2012] beschrieben.

Der Aufwand für die Aktivitäten zur Auswahl und Einführung von Testwerkzeugen ist in keinem Fall zu unterschätzen. Zum Beispiel verdreifacht die Einführung eines neuen, unbekannten Tools zur Testautomatisierung den Testaufwand der zu automatisierenden Testfälle im ersten Projekt![1]

Die Tab. 15.2 zeigt eine Übersicht der unterschiedlichen Arten von Testwerkzeugen, die im Teil II bei den jeweiligen Testarten beschrieben sind.

Neben diesen speziellen Testwerkzeugen werden weitere Tools benötigt, die bei mehreren Testarten und in jeder Teststufe eingesetzt werden können und daher nicht bei den Beschreibungen der einzelnen Testarten aufgeführt sind. Zu diesen Werkzeugen gehören:

- **Datenbank- und Texteditoren** zum Auslesen und Manipulieren von Dateien und Datenbanken
- **Komparatoren** zum Vergleichen von Datenbankinhalten und Dateien
- **Testdatengeneratoren** zum Generieren von Testdaten, eventuell aus bestehenden Datenstrukturen oder bestehendem Sourcecode
- **Tools zur Anonymisierung** von Produktionsdaten

Zudem können folgende Werkzeuge die Qualitätsmanagementprozesse in einem Projekt effizient unterstützen:

- **Anforderungsmanagementwerkzeuge** zur Verwaltung der Spezifikationsdokumente
- **Konfigurationsmanagementwerkzeuge** zur Versions- und Konfigurationsverwaltung von Systemkomponenten, Testmitteln und Dokumenten
- **Fehlermanagementwerkzeuge** zur Verfolgung von Fehlern und Erstellung von Fehlerstatistiken
- **Projektmanagementwerkzeuge** zur Planung und Überwachung der Testaktivitäten und Testressourcen

Sogenannte **Testsuiten** integrieren mehrere Werkzeugtypen in einem, d. h. unter einer einheitlichen Benutzerführung können mehrere Aufgaben erledigt werden. In einer komfortablen Testsuite können zum Beispiel die Anforderungen und Testfälle beschrieben, die Testfälle mit Risikoklasse versehen, Testszenarien zusammengestellt, Skripte zur automatischen Testdurchführung generiert und ausgeführt, Tests protokolliert und Fehler verfolgt werden.

Diese integrierten Werkzeuge sind sehr nützlich, aber sie sind nicht preiswert und ihre Einführung für ein Unternehmen bzw. für ein Projekt will wohl überlegt und geplant sein.

Übersichten zu allen Arten von Testwerkzeugen sind zu finden unter [URL: TestToolReview], [URL: TestToolsJava], [URL: TestToolsOPENSOURCE] und [URL: TestToolsSQA].

Das Testmanagement muss auch entscheiden, ob und welche Testmittel gekauft oder gemietet werden sollen. Werden Tools oder andere Testmittel nur temporär benötigt, kön-

[1] Erfahrungswerte des Autors

Tab. 15.2 Spezielle Testwerkzeuge

Kapitel	Testart	Testwerkzeug
8	Dokumententest	CASE-Tools Review-Managementwerkzeuge
9	*Tests zur Funktionalität*	
9.1	Unit-Test	Code Analyzer Treiber, Platzhalter Testframeworks (xUnit-Tools) Memory-Checker, Profiler Code Coverage Tools
9.2	Funktionaler Komponententest	Treiber, Platzhalter, Mocks Capture Replay Tools Code Coverage Tools
9.3	Integrationstest	Treiber, Platzhalter, Mocks Capture Replay Tools Monitore
9.4	Funktionaler Systemtest	Capture Replay Tools Monitore
9.5	Link-Test	Link-Checker
9.6	Cookie-Test	Browser, Browser Add-ons Cookie Viewer
9.7	Plugin-Test	Capture Replay Tools User Tracking Tools
9.8	Test der Browser-Einstellungen	Browser, Browser Add-ons
9.9	Webservice-Test	Webservice-Treiber Load Test Tools
9.10	Sicherheitstest	Security Test Tools
9.11	Interoperabilitätstest(Cross-Browser-Test, Cross-Device-Test)	Browser, Browser Add-ons Simulatoren, Emulatoren Virtuelle Maschinen Capture Replay Tools User Tracking Tools
9.12	Mobile-Funktionstest	Mobile-Browser Simulatoren, Emulatoren Virtuelle Maschinen Capture Replay Tools
10	*Tests zur Benutzbarkeit*	
10.1	Content-Test	Impressum-Generator
10.2	Oberflächentest	Capture Replay Tools User Tracking Tools
10.3	Usability-Test	Eye Tracker User Tracking Tools
10.4	Zugänglichkeitstest	Spezielle Browser Tools zur Zugänglichkeitsprüfung

Tab. 15.2 (Fortsetzung)

Kapitel	Testart	Testwerkzeug
10.5	Auffindbarkeitstest	Ranking Tools Backlink-Checker
11	*Tests zur Änderbarkeit und Übertragbarkeit*	
11.1.4	Statische Code-Analyse durch Werkzeuge	Compiler Code Analyzer Validatoren
11.2	Installationstest	Capture Replay Tools
11.2.2	Mobile-Installationstest	Simulatoren, Emulatoren Capture Replay Tools
12	*Tests zur Effizienz*	
12	Effizienztest	Load Test Tools Monitore
13	*Tests zur Zuverlässigkeit*	
13.1	Ausfallsicherheitstest	Load Test Tools Monitore
13.2	Verfügbarkeitstest	Monitore
13.3	Mobile-Zuverlässigkeitstest	Monitore

nen diese zum Beispiel bei Anbietern extern via **Test-Cloud** kostengünstiger beschafft werden.

Zum Beispiel rechnet es sich i.d. R nicht, für einen Last- oder Stresstest eine eigene globale Serverfarm zu unterhalten und viele tausend Lizenzen für virtuelle User zu kaufen. Oder alle möglichen Browser, Betriebssysteme und mobilen Endgeräte für Interoperabilitätstests vorzuhalten, können sich auch nicht viele Unternehmen leisten. Dazu mehr im folgenden Abschnitt.

15.2.5 Bereitstellung der mobilen Endgeräte

Wo kommen die Endgeräte her?

Einige der in Teil II beschriebenen Tests für Mobile-Webseiten und Mobile-Apps können auf Desktop-Rechnern durchgeführt werden. Ein paar weitere Tests können mit Simulatoren bzw. Emulatoren für mobile Geräte überprüft werden. Aber für finale Tests wie zur Funktionalität, Usability, Zuverlässigkeit und Sicherheit werden reale Tablet-PCs und Smartphones benötigt. Und das können ganz schön viele sein. Stellt sich dem Testmanager die Frage: „Wo bekommen wir die vielen Endgeräte her?“

Alle Geräte kaufen?
Als teuerste und aufwändigste Möglichkeit können alle Geräte gekauft werden. Aber wie viele Komponenten kommen monatlich neu auf den Markt?

Privatgeräte der Mitarbeiter?
Eine Lösung ist, die privaten Geräte der Mitarbeiter im Unternehmen und natürlich auch gleich die Mitarbeiter als Tester zu nutzen. Ist die Mitarbeiteranzahl im eigenen Unternehmen hinreichend groß, könnten hier schon viele unterschiedliche Systemkonfigurationen in erreichbarer Nähe sein. Mitarbeiter haben in der Regel privat mobile Endgeräte und jeder hat seine Vorlieben.

Unterschiedliche Firmengeräte?
In diesem Sinne kann es auch interessant sein, den Mitarbeitern unterschiedliche Firmengeräte zur Verfügung zu stellen. Der Erfolg, die Mitarbeiter in den Test zu involvieren, hängt sicherlich von der Größe des Unternehmens und der Qualifikation der Mitarbeiter ab. Und auch bei dieser Lösung müssen die permanent neu auf den Markt geworfenen, mobilen Geräte im Auge behalten werden.

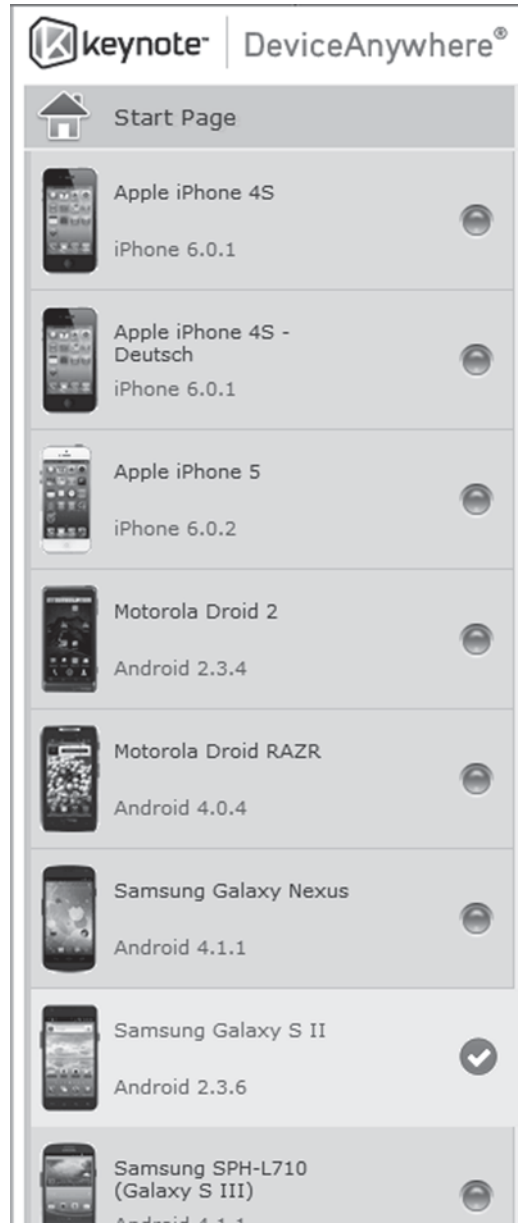
Externe Geräte?
Günstiger als die Anschaffung kann das befristete Mieten sein, was mit oder ohne Tester geschehen kann. Es gibt Anbieter von Clouds für mobile Geräte, wie zum Beispiel unter [URL: DeviceAnywhere] (s. Abb. 15.1) oder [URL: perfectomobile] zu finden. Diese Dienste stellen kostenpflichtig mobile Geräte für Testzwecke in einer Cloud zur Verfügung. Die Tester werden hier allerdings nicht „mitgeliefert".

Die erweiterte Form des „Outtestings", d. h. externes Gerät inklusive Tester, bringt uns zum Crowdtesting. Crowdtesting für unsere Mobile-Tests ist nur sinnvoll, wenn die beauftragte Test-Crowd auch die mobilen Endgeräte besitzt. Mehr dazu im Kap. 17.2.

15.3 Zusammenfassung

- Im Rahmen der Testplanung wird festgelegt, welche Testarten in welcher Ausprägung bei den Tests einer Applikation eingesetzt werden sollen.
- Alternativen zur Durchführung einer Testart werden in Abhängigkeit der vorgenommenen Bewertung ausgewählt.
- Ebenso müssen die notwendigen Testmethoden und Testmittel (Checklisten und Werkzeuge) ausgewählt, bereitgestellt und ggf. geschult werden.
- Unterschiedliche Testumgebungen für Entwicklungs- und Testteams und unterschiedliche Testphasen müssen geplant und bereitgestellt werden.
- Je nach Verwendungszweck müssen unterschiedliche Testdatenbestände geplant, erzeugt und bereitgestellt werden. Produktionsdatenbestände für Testwecke müssen anonymisiert werden.

Abb. 15.1 DeviceAnywhere



- Das Testmanagement muss entscheiden, ob und welche Testmittel gekauft oder gemietet werden sollen. Testmittel können ggf. extern kostengünstiger beschafft werden.
- Internetdienste bieten via Test-Cloud Testmittel und mobile Geräte an.
- Beim Crowdtesting besitzt die testende Test-Crowd die mobilen Endgeräte.

Planung der Teststufen

16

Wir sollten uns unter Qualitätsdruck,nicht aber unter Zeitdruck setzen.

Tyll Necker (1930–2001)

Ein Software-Entwicklungsprojekt durchläuft unter zeitlichem Aspekt mehrere Abschnitte, die im Testkonzept festgelegt und geplant werden. Unabhängig vom sequentiellen oder iterativen Vorgehen müssen Abnahmetests stattfinden und der Betrieb muss überwacht werden.

16.1 Vorgehensweisen im Projekt

Sequentielles Vorgehen
Große, langwährende Projekte werden in der Regel in sequentiell aufeinanderfolgenden Teststufen geplant und realisiert (zum Beispiel nach dem V-Modell XT[GL]), d. h. die Testobjekte werden von einer Teststufe in die nächste übergeben, sodass sich eine definierte Reihenfolge ergibt:

1. Unit-Test
2. Komponententest
3. Integrationstest
4. Systemtest
5. Abnahmetest
6. Betrieb

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_16

Die erste und die letzte aufgeführte Teststufe, der Unit-Test und der Betrieb, sind keine „offiziellen" Testphasen in den sequentiellen Modellen. Aber sie müssen bei der Testplanung berücksichtigt werden, denn Maßnahmen zur Qualitätssicherung beginnen bei der Entwicklung und sind auch während des gesamten Betriebes notwendig.

Iterativ-inkrementelles Vorgehen

Wird Software iterativ-inkrementell (z. B. mit Scrum[GL]) entwickelt, geschieht dies in kurzen Zeitintervallen von 2–4 Wochen, den Sprints. Beim Testen bestehen die Unterschiede zum sequentiellen Vorgehen darin, dass die Testaktivitäten in kleineren Einheiten, aber dafür in größerer Anzahl geplant und durchgeführt werden und dass die Testzuständigkeiten beim gesamten Team und nicht nur beim Testteam liegen.

Am Ende eines Sprints liegt stets ein funktionstüchtiges Software-System vor. Das bedeutet, dass in einem Sprint alle beschriebenen Testarten bis hin zum funktionalen und nicht-funktionalen Systemtest zum Einsatz kommen können. Bei jedem Sprint werden die bereits durchgeführten Tests immer wieder mit ausgeführt[1] und um die Testfälle und Tests der neuen Iteration ergänzt.

Obwohl in einem agilen Projekt mit jedem Sprint prinzipiell eine produktionsfähige Software bereitsteht, stellt sich die Frage, ob diese Systemkonfiguration dann auch wirklich produktiv gestellt werden kann und soll.

Continuous Integration

Der Teil der Frage nach dem „Kann" kann mit der Integrationsstrategie Continuous Integration (s. Abschn. 9.3.1) beantwortet werden. Sie zeigt sehr schnell, ob alle Tests erfolgreich waren – und nur dann ist eine Produktivstellung zu empfehlen.

Die Frage nach dem „Soll" sollte im Demo-Meeting am Ende eines Sprints beantwortet werden. In diesem Meeting präsentiert das Team dem Product Owner[GL] die Arbeitsergebnisse, der dann entscheidet, ob er das Produkt abnimmt oder nicht.

Das Testmanagement ist verantwortlich, dass die festgelegten Testarten in den geplanten Teststufen bzw. Sprints durchgeführt werden und die Testmittel zur Verfügung stehen.

Welches Entwicklungsmodell auch eingesetzt wird, die durchzuführenden Testtätigkeiten sind immer die gleichen, d. h. Testfälle und Testdurchführungen planen, spezifizieren, dokumentieren, realisieren, protokollieren und auswerten.

In jedem Fall müssen Testaktivitäten zur Abnahme durch den Auftraggeber und für den produktiven Betrieb geplant und durchgeführt werden, was Thema der beiden folgenden Kapitel ist.

[1] S. Kap. 14.2, Regressionstest

16.2 Abnahmetest

Nachdem der Systemtest bzw. bei einem System von Systemen der Systemintegrationstest erfolgreich abgeschlossen ist, muss der Auftraggeber die Freigabe der Anwendung erteilen, d. h. das Gesamtsystem abnehmen. Das muss er auch tun, wenn er in den vorangegangenen Tests miteinbezogen war.

► Mit dem **Abnahmetest** prüft der Auftraggeber, ob die beauftragte Anwendung den vertraglich vereinbarten Anforderungen entspricht und alle Abnahmekriterien erfüllt sind.

Der Abnahmetest hat das Ziel, dem Auftraggeber die Entscheidung zu ermöglichen, ob er das beauftragte System abnehmen kann oder eben nicht. Im Rahmen des Abnahmetests können die beiden Phasen Alpha- und Beta-Test eingeplant werden, die der endgültigen Freigabe einer Software vorangehen.

► Im **Alpha-Test** wird eine Vorabversion der Software vom Auftraggeber in einer Testumgebung des Software-Lieferanten getestet.

► Im **Beta-Test** wird eine Vorabversion der Software vom Auftraggeber in der Produktionsumgebung des Auftraggebers getestet.

Natürlich ist es sinnvoll und eine wichtige Aufgabe des Testmanagements, den Auftraggeber und die Endbenutzer schon vor der Endabnahme in die Testaktivitäten miteinzubeziehen, also auch schon vor dem Alpha-Test oder dem Beta-Test. Das trifft besonders auf den Usability-Test zu, weil die Akzeptanz ein kritischer Erfolgsfaktor ist.

Zu den Anforderungsbeschreibungen eines beauftragten Anwendungssystems gehört ein Katalog von **Abnahmekriterien**, gegen den der Auftraggeber die Anwendung im Abnahmetest prüft. Allein dieser Katalog ist für die Abnahme ausschlaggebend. Projektleitung und Qualitätssicherung müssen zu Projektbeginn dafür sorgen, dass ein solcher verbindlicher Kriterienkatalog mit den Qualitätsanforderungen vereinbart wird. Nur so können unnötige Diskussionen bei der Freigabe und nachträgliche Anforderungen vermieden werden.

Die Abnahme einer Anwendung erfolgt durch Tests, die der Auftraggeber in einer produktionsnahen Systemumgebung oder sogar in der Produktionsumgebung durchführt. Anhand wichtiger und „normaler“ Geschäftsvorfälle werden Funktionalität, Benutzbarkeit und Effizienz aus Sicht des Auftraggebers geprüft.

Zur Abnahme einer Software gehört unter Umständen auch die Freigabe durch den Betreiber (Rechenzentrum), der für Übertragbarkeit, Zuverlässigkeit und Ausfallsicherheit des Systems verantwortlich ist und die entsprechenden Tests durchführt.

Funktionale und systemtechnische Ausnahmesituationen bleiben in der Regel beim Abnahmetest unberücksichtigt, weil sie in den vorangegangenen Teststufen durchgeführt und – hoffentlich – nachweislich dokumentiert worden sind. Es obliegt dem Auftraggeber, neben der Durchführung seiner Testfälle diese Testdokumentation zu überprüfen und das Verhalten der Anwendung in Ausnahmesituationen stichprobenartig zu testen.

Messungen zur Verfügbarkeit können in dieser Phase für den Produktionsbetrieb aufschlussreich sein.

16.3 Betrieb

Auch für die Betriebsphase eines Anwendungssystems müssen Qualitätssicherungsmaßnahmen durchgeführt werden, wozu der Pilotbetrieb und die laufende Überwachung des Systems gehören.

16.3.1 Pilotbetrieb

Bevor eine Anwendung für alle Benutzer freigegeben wird, kann ein zeitlich begrenzter Pilotbetrieb sinnvoll sein, wenn zum Beispiel Geschäfte mit Hilfe der Anwendung abgewickelt werden. Weil das Anwendungssystem zu diesem Zeitpunkt produktiv geschaltet ist, gehört ein Pilotbetrieb zur Betriebsphase.

Im **Pilotbetrieb** wird die Anwendung in der Produktionsumgebung des Auftraggebers einem eingeschränkten und wohldefinierten Benutzerkreis (Pilotanwender) für einen begrenzten Zeitraum freigeschaltet, mit dem Ziel, Funktionalität, Benutzbarkeit und Performanz der Anwendung zu bewerten.

Wenn das neue System Teil eines umfassenderen Gesamtsystems ist, muss ein End-to-End Test durchgeführt werden. Dazu werden die wichtigsten Geschäftsprozesse vom Anfang bis zum Ende, d. h. über alle Systeme hinweg, durchgeführt.

► Der **End-to-End Test** stellt sicher, dass die Geschäftsprozesse systemübergreifend richtig und vollständig ausgeführt werden.

Wenn zum Beispiel unser Finanzierungsrechner (inklusive Autokonfigurator und Vertragsverwaltung) in einem größeren Verbund von Anwendungssystemen mit Marketingplattform, Händlervertriebsportal und SAP-Rechnungswesen betrieben wird, sollten hierfür übergreifende End-to-End Tests durchgeführt werden. Damit wird nachgewiesen, dass die Geschäftsprozesse über die Systeme hinweg einwandfrei abgewickelt werden können.

Folgende Punkte sind bei der Planung des Pilotbetriebes einer Applikation zu beachten:

✓ **Checkliste Planung Pilotbetrieb**

- Ist der Pilotbetrieb zeitlich begrenzt?
- Sind Kriterien für die Beendigung bzw. die Verlängerung des Pilotbetriebes festgelegt?
- Sind die Pilotanwender repräsentativ ausgewählt?
- Ist sichergestellt, dass nur die Pilotanwender die Anwendung erreichen können?
- Sind Geschäftsvorfälle für den End-to-End-Test festgelegt und priorisiert?
- Sind die Entwicklungs- und Testressourcen für die Pilotphase eingeplant?
- Ist ein Verfahren beschrieben, wie die von den Pilotanwendern gefundenen Fehler und eingebrachten Verbesserungsvorschläge dokumentiert, bewertet und umgesetzt werden?
- Sind die Verantwortlichen und Beteiligten für den Pilotbetrieb benannt?
- Sind Messungen der Performanz, des Ressourcenverbrauchs und der Verfügbarkeit geplant?
- Sind Usability-Umfragen und User Tracking im Pilotbetrieb geplant?
- Sind regelmäßige Feedback-Veranstaltungen mit den Pilotanwendern geplant?

Erst wenn die „Kinderkrankheiten" während der Pilotphase behoben und die Pilotbenutzer „zufrieden" sind, darf die Anwendung allen Benutzern zugänglich gemacht werden. Das setzt voraus, dass die Kriterien zur Beendigung des Pilotbetriebes als Qualitätsanforderungen festgelegt worden sind, wie zum Beispiel:

- Der Usability-Test ist positiv abgeschlossen.
- Die vereinbarte Systemverfügbarkeit wird eingehalten.
- Die maximal erlaubte Anzahl neu auftretender Fehler pro Zeiteinheit und Risikoklasse wird nicht überschritten.
- Der End-to-End-Test mit den definierten Testszenarien ist erfolgreich abgeschlossen.

16.3.2 Qualitätsprüfungen im laufenden Betrieb

Während der Pilotphase und des sich anschließenden Betriebes eines Systems zeigt sich, ob die Qualitätsanforderungen Performanz, Ressourcenverbrauch, Zuverlässigkeit, Auffindbarkeit und Wartbarkeit erfüllt werden und dauerhaft gehalten werden können. Daher sind permanente Messungen und Prüfungen im laufenden Betrieb notwendig:

- Externe Links müssen regelmäßig daraufhin geprüft werden, dass sie ihr Ziel noch erreichen und, wenn dieses der Fall ist, dass die verlinkten Inhalte sich nicht signifikant verändert haben.
- Performanz und Ressourcenverbrauch eines produktiven Systems müssen überwacht werden, damit bei Verschlechterung der Systemleistung rechtzeitig gehandelt werden kann.

- Die Verfügbarkeit und die daraus abgeleitete Ausfallrate müssen gemessen werden.
- Da im Internet immer neue Sicherheitslücken und -risiken bekannt werden und auftreten können, müssen regelmäßig die Sicherheitsmaßnahmen überprüft und die Sicherheitstests erweitert und wiederholt werden.
- Die Auffindbarkeit einer Website muss regelmäßig überprüft werden, um sicherzustellen, dass sie in Suchmaschinen dauerhaft in den vorderen Positionen zu finden ist.
- Web-Apps und Mobile-Web-Apps müssen regelmäßig den Interoperabilitätstest mit den neu auf den Markt geworfenen Systemkomponenten (Browsern, Betriebssystemen, Plugins,…) und Geräten bestehen!

16.4 Zusammenfassung

- In einem Anwendungsentwicklungsprojekt, das sequentiell vorgeht, sind die Teststufen Unit-Test, Komponententest, Integrationstest, Systemtest, Abnahmetest und Betrieb zu planen.
- Bei iterativ-inkrementeller Software-Entwicklung werden die Testaktivitäten in Sprints von 2–4 Wochen geplant und durchgeführt.
- In jeder Teststufe und in jedem Sprint kommen mehrere Testarten zum Tragen.
- Die Planung der Testressourcen ist eine komplexe Aufgabe, zumal vielfältige Abhängigkeiten zu den übrigen Projektgrößen Budget, Fertigstellungstermine, Personal und technische Ressourcen bestehen.
- Der Abnahmetest kann in mehreren Stufen mit Alpha-Test und Beta-Test durchgeführt werden. Der Alpha-Test findet beim Lieferanten statt, der Beta-Test beim Auftraggeber.
- In der Pilotphase wird eine Anwendung zum Start des Betriebes einem begrenzten Benutzerkreis zur Verfügung gestellt.
- Der End-to-End Test stellt sicher, dass die Geschäftsprozesse systemübergreifend richtig und vollständig ausgeführt werden.
- Auch im laufenden Betrieb werden Qualitätssicherungsmaßnahmen durchgeführt.
- Interoperabilitätstests müssen aufgrund des sich permanent verändernden IT-Marktes regelmäßig und intensiv durchgeführt werden.

Planung des Testteams

17

Management ist die Kunst, drei Leute dazu zu bringen, die Arbeit von drei Leuten zu tun.

William Feather (1889–1969)

Last but not least muss das Testteam zusammengestellt und zeitlich eingeplant werden. Dazu gehört sicherlich immer eine Anzahl von dezidierten Testern. Aber unter Umständen kann es sehr effektiv sein, dieses Testteam durch eine anonyme Testergruppe zu erweitern.

17.1 Das dezidierte Testteam

Ausgebildete Tester
Für die Planung und Steuerung der QS-Maßnahmen, den Entwurf von Testfällen und die Durchführung von Tests werden Tester benötigt, die in der Anwendung von Testmethoden und Testwerkzeugen ausgebildet und erfahren sind.

Testexperten
Dieses „Stammtestteam" muss temporär und aufgabenabhängig erweitert werden. Bei der Beschreibung der Testarten im zweiten Teil dieses Buches wird an mehreren Stellen darauf hingewiesen, dass für bestimmte Tests Experten eingesetzt werden sollten, um die Tests effizient durchführen zu können. Generell werden Testexperten eingesetzt, wenn fachliches oder technisches Wissen für einen Test notwendig ist oder wenn ein Testwerkzeug eine besondere Ausbildung erfordert. Einige Beispiele:

- Fachliche Dokumente werden von Fachexperten geprüft, die mit den Themen der Inhalte vertraut sind und diese beurteilen können.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5_17

- Der Content-Test wird von Rechts- und Marketingexperten vorgenommen.
- Nicht automatisierte Code-Analysen werden von Personen durchgeführt, welche die eingesetzte Programmiersprache beherrschen.
- Der Sicherheitstest muss von Sicherheitsexperten und eventuell von „legalen Hackern" durchgeführt werden.
- Für den Entwurf von technischen Testfällen, wie sie zum Beispiel für einen Plugin-Test oder den Integrationstest von SOAP-Komponenten benötigt werden, sind entsprechende Technologieexperten zuständig.
- Capture Replay Tools, Load Test Tools und Monitore werden von Testern eingesetzt, die dafür ausgebildet sind, denn diese Werkzeuge verlangen spezielle Programmierkenntnisse.
- SEO-Fachleute überprüfen die Qualität des Programmcodes in Bezug auf seine Optimierung für Suchmaschinen.
- Der Einsatz von Page Ranking Tools und User Tracking Tools verlangt Erfahrungen im Web-Marketing.

Neutrale Tester
Nicht nur Testspezialisten müssen zum Testen eingeplant werden. Zum Beispiel können Personen ohne Testausbildung den Oberflächentest durchführen oder neutrale Personen für den Gebrauchstauglichkeitstest im Usability-Labor rekrutiert werden.

Auftraggeber als Tester
Für den Abnahmetest und für Prüfungen von wichtigen Zwischenergebnissen müssen Fachleute vom Auftraggeber benannt werden und zum Zeitpunkt der Prüfung bereitstehen. Dieser Personenkreis muss, wie alle anderen am Test beteiligten Personen auch, mit ihren zu leistenden Aufwänden und den festgelegten Terminen in die Testplanung aufgenommen werden.

17.2 Die Test-Crowd

Unter Umständen kann es sehr effektiv sein, das dezidierte Testteam durch ein anonymes Testteam – eine **Test-Crowd** – zu erweitern, um die „Schwarmintelligenz" des Internets zu nutzen.

► Ein **Crowdtest** ist das Testen von Software über das Internet durch eine große Anzahl von Testern.

Crowdtesting bietet sich an für Cross-Browser-Tests, Cross-Device-Tests, Usability-Tests und Sicherheitstests. Herausforderungen an das Testmanagement beim Crowdtest einer Mobile-App sind die Koordination der zu testenden Endgeräte, der durchzuführenden Testfälle, der Testprotokollierung und des Fehlermanagements.

Es muss festgelegt werden, ob und welche Testfälle der Test-Crowd bereitgestellt werden oder ob die Crowd anhand einer Test-Charta[1] testen soll. Eine Kommunikationsplattform für Testfälle, Fehlerbeschreibungen und Fehlernachtests muss der Crowd bereitgestellt werden.

Eine Liste von Anbietern von Crowdtests ist unter [URL: crowdtesting] zu finden.

17.3 Zusammenfassung

- Das Testteam wird so zusammengestellt, dass alle fachlichen und technischen QS-Maßnahmen mit dem notwendigen Know-how durchgeführt werden können.
- Zum Testteam gehören ausgebildete Tester, Testexperten, neutrale Testnutzer und Fachleute des Auftraggebers.
- Eine Test-Crowd kann im Crowdtest eine App mit vielen unterschiedlichen Endgeräten und Systemkonfigurationen testen.

[1] Test-Charta s. Kap. 3.1.

Nachwort

Fehler werden gemacht, damit danach eine Erfahrung stattfindet und aus dieser wiederum das Erkennen stattfindet und dadurch eine Veränderung vorgenommen wird.

Konfuzius (551– 479 vor Christus)

Schneller als UMTS ist HSDPA (High Speed Downlink Packet Access). Windows 8.1 ist auf dem Markt. Die Zukunft des mobilen Internets ist Long Term Evolution (LTE). Dolphin und Skyfire sind mobile Browser, die noch nicht in dem Buch erwähnt sind.

In diesem Moment, in dem ich das Buch abschließe, gibt es schon wieder neue Technologien, neue Endgeräte, neue Testwerkzeuge und vielleicht auch neue Testmethoden.

Mit diesem Wissen ziehe ich einen Schlussstrich unter die zweite Auflage dieses Buches, und hoffe, dass die dritte nicht erst wieder in sieben Jahren erscheint.

Bis dahin
Klaus Franz

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5

Synonyme

Änderbarkeit = Wartbarkeit
Anforderungen = Requirements
Anwendung = Applikation
Ausfallrate = Unverfügbarkeit
Accessibility = Barrierefreiheit = Zugänglichkeit
Capture Replay Tool = Testroboter
Gebrauchstauglichkeit = Usability
Komponententest = Modultest = Unit-Test
Maß = Metrik
Performanz = Zeitverhalten
Risikoklasse = Risikostufe
Testart = Testtyp
Testfallentwurfsverfahren = Testentwurfsverfahren
Testmethode = Testverfahren
Teststufe = Testphase
Portabilität = Übertragbarkeit
Strukturtest = Whitebox-Test
Software-Anwendung = Software-Applikation
zustandsbasierter Test = zustandsbezogener Test
Zustandsdiagramm = Zustandsübergangsdiagramm

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5

Glossar

.NET
.NET ist eine Software-Technologie von Microsoft. Sie stellt eine virtuelle Laufzeitumgebung sowie ein Rahmenwerk von Klassenbibliotheken und Diensten für die Software-Entwicklung zur Verfügung.

Active Server Pages.NET (ASP.NET)
ASP.NET (Nachfolger der Webtechnologie ASP) ist eine serverseitige Technologie von Microsoft® zur Entwickelung dynamischer Webseiten, Webanwendungen und Webservices auf Basis von.NET.

ActiveX
ActiveX ist ein Software-Komponentenmodell von Microsoft, das es ermöglicht, aktive Objekte (Video, Sound, …) in Dokumente wie zum Beispiel Webseiten einzubauen.

Adobe Integrated Runtime (AIR)
AIR von Adobe Systems ist eine plattformunabhängige Laufzeitumgebung zur Erstellung von Rich Internet Applications (RIA) für den Desktop. AIR-Web-Apps können direkt aus dem Web installiert werden und laufen ohne Webbrowser und auf dem Desktop. Aktuell werden die Betriebssysteme Android, BlackBerry Tablet OS, BlackBerry 10, iOS, Mac OS X und Windows unterstützt.

Agile Software-Entwicklung
Agile Software-Entwicklung ist eine auf iterativer und inkrementeller Entwicklung basierende Gruppe von Software-Entwicklungsmethoden, wobei sich Anforderungen und

In diesem Glossar sind Begriffe aufgenommen, die nicht im Text des Buches definiert und dort mit [GL] gekennzeichnet sind.

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5

Lösungen durch die Zusammenarbeit von selbstorganisierenden, funktionsübergreifenden Teams entwickeln.

AJaX (Asynchronous JavaScript and XML)
AJaX ist eine Technik, mit der bei aktiviertem JavaScript Daten vom Server nachgeladen werden können, ohne dass die gesamte Seite neu geladen werden muss. Durch die asynchrone Datenübertragung werden die Wartezeiten für den Benutzer minimiert.

Apache Flex
Apache Flex (früher Adobe Flex, jetzt Apache-Projekt) ist ein Webentwicklungs-Framework, mit dem sich plattformunabhängige Rich Internet Applications (RIAs) entwickeln lassen, die im Adobe Flash Player ablaufen.

Breadcrumb
Breadcrumbs (engl. Brotkrumen) sind Navigationselemente, die den Pfad zur aktuellen Webseite anzeigen. Eine Breadcrumb-Navigationsleiste verbessert die Orientierung innerhalb verzweigter Websites und bietet Links zu den vorher besuchten, übergeordneten Webseiten an.

Clustered Server
Ein Clustered Server ist die Kopplung mehrerer Server zu einem virtuellen Server. Dadurch erhöht sich sowohl die Leistungsfähigkeit als auch die Ausfallsicherheit des Systems.

Cookie
Ein Cookie (engl. Keks) ist eine kleine Datei, die lokal auf dem Rechner des Nutzers einer Webanwendung abgelegt wird und in der Informationen abgespeichert werden, die im Zusammenhang mit der aktuellen Website stehen.

In einem Cookie sind Nutzdaten und Informationen darüber abgelegt, wer das Cookie gesetzt hat und wie lange es gültig bleiben soll. Cookies haben eine Lebensdauer von mehreren Tagen oder Wochen. Sogenannte Session-Cookies sind nur so lange aktiv, wie der Browser geöffnet ist.

Ein Cookie kann nur von Servern ausgelesen werden, die den gleichen Domain-Namen haben wie der Server, der das Cookie geschrieben hat. Ein „normales“ Cookie kann maximal 4 KB groß sein.

Common Object Model (COM)
COM ist Microsofts Software-Architektur zur Kommunikation von Software-Komponenten.

Cross Site Scripting (XSS)
Cross Site Scripting bezeichnet das Ausnutzen einer Sicherheitslücke in Webanwendungen, indem Informationen aus einem Kontext, in dem sie nicht vertrauenswürdig sind,

in einen anderen Kontext eingefügt werden, in dem sie als vertrauenswürdig eingestuft sind. Aus diesem vertrauenswürdigen Kontext wird dann ein Angriff gestartet. Ein Ziel ist, sensible Daten des Benutzers zu bekommen, wie z. B. Session-Cookies oder Tastatureingaben.

Cascading Style Sheet (CSS)
Cascading Style Sheets erweitern durch spezielle Formatanweisungen die Möglichkeiten von HTML, Websites zu gestalten. Sie ermöglichen die Trennung von Inhalt und Layout bei der Programmierung von Webseiten. Text- und Darstellungsattribute können in Klassen zusammengefasst und mit HTML-Seiten verknüpft werden.

Daemon (Disk And Execution MONitor)
Ein Daemon ist ein Programm in einem Netzwerk, das für den Benutzer unsichtbar im Hintergrund auf bestimmte Ereignisse wartet. Tritt ein solches Ereignis ein, wird eine bestimmte Aktion des Daemons ausgelöst. Zum Beispiel bearbeitet ein Mailer-Daemon im Hintergrund den Eingang von E-Mails.

Domain Name System (DNS)
DNS ist ein Dienst im Internet und anderen TCP/IP-Netzen, der den Namensraum im Internet verwaltet, d. h. für einen Hostnamen die entsprechende IP-Adresse zurückgibt und umgekehrt.

Emulator
Ein Emulator ist ein Gerät, Computerprogramm oder System, das die gleichen Eingaben akzeptiert und die gleichen Ausgaben erzeugt wie ein gegebenes System [nach IEEE 610].

Ein Emulator ist immer Software-basiert und kann unter Umständen auf einer zusätzlichen Emulations-Hardware installiert sein. Ein Emulator versucht, die Funktionen eines anderen Systems möglichst vollständig, d. h. 100-prozentig nachzuahmen. In unserem Mobile-Kontext ist ein Emulator eine Desktop-Anwendung, die ein mobiles Endgerät inklusive Betriebssystem nachahmt. Emulieren bedeutet genau nachbilden.

Gateway
Ein Gateway ist eine Hard- und/oder Softwarelösung, die eine Verbindung von inkompatiblen Netzwerken schafft.

Hosts-Datei
Die Hosts-Datei ist eine lokale Textdatei auf dem Client, in der Hostnamen festen IP-Adressen zugeordnet werden.

Hyperlink
Ein Hyperlink ist die Verbindung eines Elements eines Dokumentes zu einer anderen Stelle im selben Dokument oder zu einem anderen Dokument.

Instrumentieren
Instrumentieren ist das (Werkzeuggestütztes) Einfügen von Protokoll- oder Zählanweisungen in den Quellcode eines Testobjekts, um während der Ausführung Informationen über das Programmverhalten zu sammeln. Damit lässt sich beispielsweise die Codeüberdeckung messen.

Java Database Connectivity (JDBC)
JDBC ist eine Sammlung von Klassen, die es Java-Applikationen ermöglicht, mit Hilfe von relationalen Datenbankobjekten und den entsprechenden Methoden auf Datenbanken zuzugreifen.

JavaFX
JavaFX von Oracle® ist ein Framework für plattformübergreifende Rich Internet Applications[GL] (RIAs). JavaFX läuft auf Desktop-PCs und mobilen Geräten mit einer Java-Runtime-Umgebung.

Java-Applet
Ein Java-Applet ist ein in eine Webseite eingebettetes Java-Programm. Es wird vom Server auf den Client übertragen und dort innerhalb des Web-Browsers ausgeführt.

Java-Archiv
Ein Java-Archiv (JAR-Datei, Dateiendung „.jar“) ist ein plattform-unabhängiges Dateiformat, in dem verschiedene Dateien zusammengefasst werden können, die zur Ausführung eines Java-Applets nötig sind.

Koexistenz
Die Fähigkeit einer Software mit anderer Software in einer gemeinsamen Umgebung die gemeinsamen Ressourcen zu teilen.

Lateinisches Quadrat
Ein lateinisches Quadrat L ist eine quadratisches Matrix mit n Zeilen und n Spalten; dabei ist jedes Feld mit einer Zahl von 1 bis n belegt, sodass jede Zahl von 1 bis n in jeder Zeile und in jeder Spalte jeweils genau einmal auftritt. n ist eine natürliche Zahl und wird Ordnung des lateinischen Quadrats genannt. Ein lateinisches Quadrat L_4 ist z. B.

Load Balancer
Ein Load Balancer verteilt die anfallende Last auf mehrere Webserver, die zu einem virtuellen Server zusammengeschlossen sind. Load Balancer existieren in Form von Hardware, Software und als Kombination von beidem. Sie werden paarweise eingesetzt, damit sie nicht selbst zur Schwachstelle im System werden.

Ordnungsmäßigkeit (Compliance)
Ordnungsmäßigkeit einer Software, ist die Eigenschaft, anwendungsspezifische Normen oder Vereinbarungen oder gesetzliche Bestimmungen und ähnliche Vorschriften zu erfüllen.

Portable Document Format (PDF)
PDF ist ein plattformübergreifendes Dateiformat für Dokumente, das von der Firma Adobe Systems entwickelt wurde.

Product Owner
Der Product Owner ist für die strategische Entwicklung eines Produktes zuständig. Er verantwortet die Spezifikation und die Priorisierung der jeweils zu entwickelnden Produkteigenschaften. Der Product Owner allein entscheidet, ob die vom Entwicklungsteam am Ende jedes Sprints gelieferte Anwendung akzeptabel ist und ob und wann die Produktivstellung der Anwendung erfolgt.

Proxy-Server
Proxy-Server dienen als Schnittstelle zwischen dem Internet und dem lokal eingesetzten Rechner. Zum einen sorgen sie für schnellere Datenbereitstellung, weil sie eine einmal aufgerufene Datei zwischenspeichern und sie bei einem erneuten Aufruf schneller bereitstellen. Zum anderen erhöhen sie die Sicherheit, denn die im Internet verbundenen Komponenten wissen zum Beispiel nicht, mit welchem Rechner aus dem internen Netzwerk sie verbunden sind, weil sie nur die Adresse des Proxy-Servers kennen.

QR-Code
Der QR-Code (QR = Quick Response) ist ein Standard für die Darstellung von Informationen mit Hilfe einer zweidimensionalen Grafik. Er wurde 1994 von der japanischen Firma Denso Wave entwickelt, um damit die Logistik zu unterstützen. Der Code wurde 2000 von der ISO unter ISO/IEC 18004 standardisiert.

Real Time Messaging Protocol (RTMP)
RTMP von Adobe Systems ist ein Netzwerkprotokoll, mit dem u. a. Audio- und Video-Daten von einem Server zu einem Flash-Player auf dem Client übertragen werden.

Relay-Host
Ein Relay-Host ist ein wesentlicher Bestandteil von Firewall-Sicherheitssystemen. In einem durch Firewalls geschützten Netz ist der Relay-Host der einzige Host, der eine Verbindung zum Internet herstellen kann.

Rich Internet Application (RIA)
Rich Internet Applications erlauben „reichhaltige“ Darstellungs- und Interaktionsmöglichkeiten für den Benutzer. RIAs können in Webbrowsern laufen, zum Beispiel unterstützen

sie dort diverse Medienformate und erlauben dem Benutzer „Drag&Drop“-Funktionalitäten. RIAs können aber auch eigenständig auf dem Client laufen, d. h. nicht zwangsläufig im Browser.

Roaming
Roaming ist ein technisches Übergabeverfahren zwischen zwei Funkzellen. Es ermöglicht eine unterbrechungsfreie Mobilkommunikation, wenn ein mobiles Gerät den Funkbereich einer Funkzelle verlässt und in den einer benachbarten Funkzelle eintritt.

Roaming kann auch grenzüberschreitend erfolgen, wenn der eigene Netzbetreiber ein entsprechendes Abkommen mit dem Netzbetreiber im Ausland hat.

Rollover
Beim Rollover wird eine Grafik gegen eine andere Grafik ausgetauscht, wenn der Mauszeiger über die Grafik fährt, zum Beispiel bei Menüpunkten.

Scrum
Scrum (engl. Gedränge) ist ein iterativ inkrementelles Vorgehensmodell für Software-Entwicklung und Projektmanagement und gehört zu den Vorgehensweisen der agilen Software-Entwicklung[GL]. Im Mittelpunkt steht das selbstorganisierte Entwicklerteam ohne den klassischen Projektleiter. Die Software wird in Zyklen von 2–4 Wochen (Sprints) entwickelt und getestet, in denen das Scrum-Team „ungestört“ arbeitet.

Simulator
Ein Simulator ist ein Gerät, Computerprogramm oder Testsystem, das sich wie ein festgelegtes System verhält, wenn man es mit einem definierten Satz kontrollierter Eingaben versorgt [nach IEEE 610, DO178b].

Ein Simulator versucht nicht wie ein Emulator, die Funktionen eines anderen Systems 100-prozentig nachzuahmen. In unserem Mobile-Kontext ist ein Simulator eine Anwendung, die das Verhalten eines mobilen Endgeräts nachstellt, aber weder Hardware noch Betriebssystem nachahmt. Simulieren bedeutet vortäuschten.

Sitemap
Unter einer Sitemap versteht man eine besondere Übersicht der Navigationsstruktur einer Website, zum Beispiel in der Darstellung des Windows-Explorers.

Simple Object Access Protocol (SOAP)
SOAP definiert als offizieller W3C-Standard einen Nachrichtenaustausch zwischen zwei Partnern im Internet.

Swing (Java-Swing)
Swing ist eine von Sun Microsystems für Java entwickelte Schnittstelle zur vereinfachten Programmierung von grafischen Benutzeroberflächen.

Telekommunikationsgesetz (TKG)
Zweck dieses Gesetzes ist es, durch Regulierung im Bereich der Telekommunikation den Wettbewerb zu fördern und flächendeckend angemessene und ausreichende Dienstleistungen zu gewährleisten sowie eine Frequenzordnung festzulegen.

Silverlight
Silverlight von Microsoft® ist eine Erweiterung für Webbrowser zur Ausführung von Rich Internet Applications[GL]. Zudem wird Silverlight als Framework für Apps für Windows Phone 7 verwendet. Silverlight ist als Plugin für Windows und Apple Macintosh verfügbar und läuft in den Browsern Internet Explorer, Firefox, Opera, Google Chrome und Safari.

Thread
Ein Thread (engl. Faden) ist ein Programmteil, der zeitlich unabhängig von anderen Programmteilen abläuft.

Transaktion
Eine Transaktion ist ein ergebnisorientierter Kommunikations- oder Verarbeitungsabschnitt. Er besteht i. d. R. aus einer Folge von Nachrichten oder Operationen, die entweder alle übermittelt bzw. ausgeführt werden oder alle auf den Zustand vor dem Beginn der Transaktion zurückgesetzt werden. Das System ist vor Beginn und nach Ende einer Transaktion in einem konsistenten Zustand.

V-Modell XT
Das V-Modell XT [URL: V-Modell] ist ein Vorgehensmodell für Software-Entwicklungsprojekte unter Berücksichtigung des gesamten Systemlebenszyklus. Das V-Modell XT definiert die in einem Projekt zu erstellenden Ergebnisse, beschreibt die Vorgehensweisen, wie diese Ergebnisse erarbeitet werden, und es legt die Verantwortlichkeiten der einzelnen Projektbeteiligten fest.

„V“ steht nicht für „Vorgehen“, sondern für die visuelle Darstellung als V, in der die einzelnen Entwicklungsstufen den entsprechenden Teststufen gegenübergestellt werden.

„XT“ steht für eXtreme Tailoring, um die hohe Anpassungsfähigkeit des V-Modells an die unterschiedlichsten Anforderungen der verschiedenen Projekte und Organisationen auszudrücken.

Web-Archiv
Ein Web-Archiv (WAR-Datei, Dateiendung „.war“) ist eine Datei im JAR- bzw. ZIP-Format, die eine vollständige Java-Web-Anwendung enthält.

Web Services Description Language (WSDL)
Die Web Services Description Language (WSDL) ist eine plattform-, programmiersprachen- und protokollunabhängige Beschreibungssprache für Webservices zum Austausch

von XML-basierten Nachrichten. Eine WSDL-Datei beschreibt die von außen aufrufbaren Funktionen eines Webservices sowie deren Ein- und Rückgabeparameter.

XHTML (Extensible Hypertext Markup Language)
XHTML ist eine Auszeichnungssprache für das World Wide Web. Das „X“ steht für „Extensible“, weil XHTML-Dokumententypen auf XML (Extensible Markup Language) basierend erweiterbar sind.

XML (Extensible Markup Language)
Extensible Markup Language engl. („erweiterbare Auszeichnungssprache“) ist ein Standard zur Erstellung maschinen- und menschenlesbarer Dokumente in Form einer Baumstruktur, der vom World Wide Web Consortium (W3C) definiert wird. XML wird u. a. für den plattform- und implementationsunabhängigen Austausch von Daten zwischen Computersystemen eingesetzt.

X-Windows
X-Windows ist ein Standard zum Betrieb von Grafikbildschirmen.

Quellen

Literatur

[BSI-SecureNet_2006] Sicherheit von Webanwendungen Maßnahmenkatalog und Best Practices, im Auftrag des Bundesamtes für Sicherheit in der Informationstechnik, erstellt von SecureNet, Version 1, August 2006

[Dustin_2001] Dustin, E.: Orthogonally Speaking–A method for deriving a suitable set of test cases, STQE Publishing, a division of Software Quality Engineering, STQE magazine September/ October 2001

[Bath_2011] Graham Bath; Judy McKay: Praxiswissen Softwaretest - Test Analyst und Technical Test Analyst: Aus- und Weiterbildung zum Certified Tester, dpunkt.verlag, 2. Auflage, Heidelberg, 2011

[Hörmann_2006] Hörmann, K.; Dittmann, L.; Hindel, B.; Müller, M.: SPICE in der Praxis - Interpretationshilfe für Anwender und Assessoren, dpunkt.verlag, 1. Auflage, Heidelberg, 2006

[Kneuper_2007] Kneuper Dr., R.: CMMI - Verbesserung von Softwareprozessen mit Capability Maturity Model Integration; dpunkt.verlag, 3. Auflage, Heidelberg, 2007

[Liggesmeyer_2009] Liggesmeyer, P.: Software-Qualität: Testen, Analysieren und Verifizieren von Software; Spektrum Akademischer Verlag; Auflage: 2. Aufl. 2009

[Myers_1987] Myers, G. J.: Methodisches Testen von Programmen: 2. Auflage, München Wien, R. Oldenbourg Verlag, 1987

[Schweier_QF-Test_2013] Dirk O. Schweier: Praxisbeispiel: datengetriebene und schlüsselwortgetriebene Testfalldarstellung, http://www.muthpartners.de/fileadmin/fachwissen/Praxisbeispiel_Testfalldarstellung.pdf, Wiesbaden, 2013

[Spillner_2012] Spillner, A.; Linz, T.: Basiswissen Softwaretest; dpunkt.verlag, 5. Auflage, Heidelberg, 2012

[Spillner_2011] Spillner, A.; Roßner, T.; Winter, M.; Linz, T.: Praxiswissen Softwaretest - Testmanagement; dpunkt.verlag, 3. Auflage, Heidelberg, 2011

[SOGETI_2011] TPI NEXT® - Geschäftsbasierte Verbesserung des Testprozesses, SOGETI, dpunkt.verlag, 1. Auflage, Heidelberg, 2011

[Westphal_2005] Westphal, F.: Testgetriebene Entwicklung mit JUnit & FIT - Wie Software änderbar bleibt, dpunkt.verlag, 1. Auflage, Heidelberg, 2005

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5

Normen und Standards

[DIN 60812] DIN EN 60812: 2006-11 Analysetechniken für die Funktionsfähigkeit von Systemen - Verfahren für die Fehlzustandsart- und -auswirkungsanalyse (FMEA)
[DIN 40041] DIN 40041: 1990-12 Zuverlässigkeit; Begriffe
[DIN 66272] DIN 66272: 1994-10 Bewerten von Softwareprodukten - Qualitätsmerkmale und Leitfaden zu ihrer Verwendung
[IEEE 829] IEEE Std 829: 2008 IEEE Standard for Software and System Test Documentation
[IEEE 1219] IEEE 1219: 1998 Standard for Software Maintenance, Institute of Electrical and Electronics Engineers, 1998
[ISO/IEC 25000] ISO/IEC 25000:2005-08 Software-Engineering - Qualitätskriterien und Bewertung von Softwareprodukten (SQuaRE) - Leitfaden für SQuaRE
[ISO 9241-11] DIN EN ISO 9241-11: 1998 Ergonomische Anforderungen für Bürotätigkeiten mit Bildschirmgeräten - Teil 11: Anforderungen an die Gebrauchstauglichkeit - Leitsätze
[ISO 9241-110] DIN EN ISO 9241-110: 2008-09 Ergonomie der Mensch-System-Interaktion - Teil 110: Grundsätze der Dialoggestaltung
[ISO 9241-151] DIN EN ISO 9241-151: 2008 Ergonomie der Mensch-System-Interaktion - Teil 151: Leitlinien zur Gestaltung von Benutzungsschnittstellen für das World Wide Web
[ISO 9241-171] DIN EN ISO 9241-171: 2008 Ergonomie der Mensch-System-Interaktion - Teil 171: Leitlinien für die Zugänglichkeit von Software
[ISTQB_2013] ISTQB®/GTB Standardglossar der Testbegriffe Deutsch/Englisch, Version 2.2, Ausgabestand: 19. April 2013, erstellt von der GTB Working Party Glossary

URLs

[URL: AbITools] http://www.wob11.de/programme.html Werkzeuge zum Zugänglichkeitstest, Aktionsbündnis für barrierefreie Informationstechnik (AbI)
[URL: ALLPAIRS] http://www.satisfice.com/tools.shtml ALLPAIRS Test Case Generation Tool, James Bach, Satisfice, Inc.
[URL: AM-Tools] http://de.wikipedia.org/wiki/Anforderungsmanagement-Software Liste von Anforderungsmanagementwerkzeugen
[URL: ArgoUML] http://argouml.tigris.org/ Tigris.org, Open Source Software Engineering Tools
[URL: Backlinktest] http://www.backlinktest.com/ Backlink Checker von Maik Wildemann & Sebastian Schöne, 2.0Promotion GbR, Leipzig
[URL: BacklinkChecker] http://www.karlkratz.de/onlinemarketing-blog/backlink-checker-tools/ Eine Beschreibung empfehlenswerter Backlink Checker Tools von karl, kratz onlinemarketing, Berlin
[URL: BaNu] http://www.banu.bund.de BaNu - Barrieren finden, Nutzbarkeit sichern, Anwendung zur Prüfung von E-Government Angeboten des Bundesamtes, Köln
[URL: Beuth] http://www.beuth.de/ Beuth Verlag GmbH
[URL: BGG] http://www.wob11.de/bgg.html Auszüge aus dem Behindertengleichstellungsgesetz des Bundes (BGG), Web ohne Barrieren nach Paragraph 11 im Bundesbehindertengleichstellungsgesetzes, vom Aktionsbündnis für barrierefreie Informationstechnik (AbI)

Die in diesem Buch aufgeführten URLs wurden mit Drucklegung überprüft. Eine Garantie für die Gültigkeit über dieses Datum (Mai 2014) hinaus kann nicht übernommen werden.

[URL: BITV] http://www.gesetze-im-internet.de/bitv_2_0/ Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz
[URL: BITV-Lotse] http://www.bitv-lotse.de Lotse durch die Barrierefreie Informationstechnik-Verordnung (BITV) 2.0,
[URL: BITVCL] http://www.wob11.de/checklisten.html Checklisten zur Barrierefreien Informationstechnik Verordnung (BITV), Web ohne Barrieren nach Paragraph 11 im Bundesbehindertengleichstellungsgesetzes, vom Aktionsbündnis für barrierefreie Informationstechnik (AbI)
[URL: Browser-Archiv] http://browsers.evolt.org/ Browser-Archiv von evolt.org
[URL: BrowserSandbox] https://spoon.net/browsers Dienst zum Cross-Browser-Test in virtuellen Umgebungen von Spoon.net, Seattle
[URL: BrowserSnapshot] http://browsershots.org/ Online-Dienst, erzeugt Screenshots von Webseiten mit unterschiedlichen Browsern auf unterschiedlichen Betriebssystemen von Browsershots.org
[URL: Browser-Statistik] http://www.browser-statistik.de/ Browser-Statistik.de, ein Projekt von Jan Papenbrock
[URL: BSI] http://www.bsi.bund.de Bundesamt für Sicherheit in der Informationstechnik (BSI)
[URL: CaseMaker] http://www.casemaker.de/ CaseMaker, Testspezifikationswerkzeug, Díaz & Hilterscheid Unternehmensberatung GmbH, Berlin
[URL: CASE-Tools] http://de.wikipedia.org/wiki/Computer-Aided_Software_Engineering CASE-Tool-Übersicht, Wikipedia - Stichwort „Computer-Aided Software Engineering"
[URL: CTE XL] http://www.berner-mattner.com/en/berner-mattner-home/products/cte/index.html, Klassifikationsbaum-Editor CTE XL (ab Sommer 2014 wird das Tool Testona heißen), Berner & Mattner Systemtechnik GmbH, München
[URL: CombTest] http://alarcosj.esi.uclm.es/CombTestWeb/combinatorial.jsp Combinatorial testing page
[URL: crowdtesting] http://www.crowdtesting.biz Website zum Crowd Testing von Stefan Baader, München
[URL: denic] http://denic.de Zentrale Registrierungsstelle für alle Domains unterhalb der Top Level Domain.de, DENIC eG, Frankfurt am Main
[URL: DeviceAnywhere] http://deviceanywhere.com/ DeviceAnywhere® ist eine Plattform für den Test mit mobile Geräten in einer Cloud, Produkt von Keynote Systems, Inc.®, San Mateo, California
[URL-DiJi-Tools] http://di-ji.de/r/testtools Aufstellung von Testwerkzeugen für Zugänglichkeitstests, Digital informiert - im Job integriert - Di-Ji, ein vom Bundesministerium für Arbeit und Soziales (BMAS) gefördertes Projekt
[URL: DS-Webanalyse01] http://www.onlinemarketing-praxis.de/web-controlling/web-analytics-datenschutzkonform-einsetzen Informationen zur Web-Analyse von Onlinemarketing-Praxis, Asendorf
[URL: DS-Webanalyse02] http://www.bvdw.org/medien/bvdw-veroeffentlicht-kostenfreies-whitepaper-zur-datenschutzkonformen-webanalyse?media=*4007*, Whitepaper zur datenschutzkonformen Webanalyse, Bundesverband Digitale Wirtschaft (BVDW) e. V., Düsseldorf
[URL: Emulatoren] http://www.mobilexweb.com/emulators Übersicht Mobile-Emulatoren/Simulatoren, Maximiliano Firtman
[URL: Ergo] http://www.ergo-online.de/site.aspx?url=*html/software/titel.htm* Informationen zu ergonomischer Software, ergo-online®, Gesellschaft Arbeit und Ergonomie - online e. V.
[URL: GesetzeBMJ] http://www.gesetze-im-internet.de/ Gesetze in Internet, Das Bundesministerium der Justiz mit der juris GmbH
[URL: Ghostery] http://www.ghostery.com/ Visualisierung von Tracking-Diensten, von EVIDON Inc., New Yourk

[URL: heise-XSS] http://heise.de/-170817 XSS-Schwachstelle auf Bundesregierung.de, auf **heise online,**
[URL: Hexawise] http://hexawise.com/ webbasiertes Online-Tool zum Paarweisen Testen, Hexawise, Inc.
[URL: JMeter] http://jakarta.apache.org/jmeter/ Apache JMeter, Load Test Tool, The Apache Jakarta Project - The Apache Software Foundation
[URL: JUnit] http://www.junit.org JUnit, Testframework, JUnit.org
[URL: JuraForum] http://www.juraforum.de/impressum-generator/ Disclaimer-Muster und Impressum-Generator auf JuraFurom.de, Einbock GmbH, Hannover
[URL: LFDN-Firewall] http://www.lfd.niedersachsen.de/download/31977/Orientierungshilfe_Grundschutz_durch_Firewall_LfD_Niedersachsen.pdf, Der Landesbeauftragte für den Datenschutz Niedersachsen, Orientierungshilfe "Grundschutz durch Firewall", 1999
[URL: LoadUI] http://www.loadui.org/ Open Source Tool für Lasttest von Webservices, SmartBear Software Inc. (siehe auch [URL: SoapUI])
[URL: Lynx] http://lynx.browser.org/ Lynx, textbasierter Webbrowser
[URL: Mockito] https://code.google.com/p/mockito/ Mocking-Framework, Google Project Hosting
[URL: NeoLoad] http://www.neotys.de/product/overview-neoload.html Lasttesttool für Weblösungen von Neotys, Gemenos, Frankreich
[URL: OATS] http://www.51testing.com/ddimg/uploadsoft/20090113/OATSEN.pdf Orthogonal Array Testing Strategy (OATS), Artikel von Jeremy M. Harrell
[URL: OpenVAS] http://www.openvas.org/index-de.html, Framework aus mehreren Diensten und Werkzeugen für Schwachstellen-Scanning und Schwachstellen-Management, OpenVAS Webseiten werden von der Intevation GmbH bereitgestellt
[URL: OrthoArray-1] http://www.york.ac.uk/depts/maths/tables/orthogonal.htm The University of York, Department Mathematics, Orthogonal Arrays (Taguchi Designs)
[URL: OrthoArray-2] http://testcover.com/pub/background/cover.htm On the Construction of Orthogonal Arrays and Covering Arrays Using Permutation Groups, George Sherwood
[URL: OWASP] http://www.owasp.org The Open Web Application Security Project (OWASP)
[URL: PageSpeed] https://developers.google.com/speed/pagespeed/ Tool zur Analyse der Performanzeigenschaften von Webseiten, Google
[URL: Pairwise] http://www.pairwise.org/ Pairwise Testing, Combinatorial Test Case Generation, maintained by Jacek Czerwonka
[URL: PluginDoc] http://plugindoc.mozdev.org/de-DE/ Firefox-Plugins, Deutsche Seite von PluginDoc, mozdev.org
[URL: Plugin-Java] http://www.oracle.com/technetwork/java/index-141825.html Oracle Network Technology, Oracle®
[URL: PMD] http://pmd.sourceforge.net/ Source Code Analyser für Java, JavaScript, XML, XSL, sourceforge
[URL: QCHESS] http://www.q-chess.de Q-Chess - Checklistenverwaltungssystem, G. Muth Partners GmbH, Wiesbaden
[URL: QF-Test] http://www.qfs.de Testautomatisierungs-Tool für Java und Web, Quality First Software GmbH, Geretsried
[URL: Ranorex] http://www.ranorex.de Testautomatisierungs-Tool für Desktop,- Web- und mobile Anwendungen, Ranorex GmbH, Graz
[URL: ResponsiveViewer] http://www.active-value.de/responsive-design-viewer/ „Responsive Design Viewer“ zum Prüfen responsiver Websites, active value GmbH
[URL: Screenfly] http://quirktools.com/screenfly/ QuirkTools von Kyle Schaeffer, Virginia
[URL: SCRUM] https://www.scrum.org/ Offizielle Website der Scrum.org, Boston, Massachusetts (USA)

[URL: Selenium] http://docs.seleniumhq.org/ Selenium, Capture Replay Tool für Webapplikationen, OpenQA
[URL: SEO-Google] http://www.google.com/intl/de/webmasters/guidelines.html Richtlinien für Webmaster zu Suchmaschinenoptimierung, Google
[URL: SenSEO] http://www.sensational-seo.com/ Analysetool für Webseiten auf SEO-Kriterien von Nico Steiner
[URL: SoapUI] http://www.soapui.org/ Open Source Tool insbesondere für Funktionstest von Webservices, SmartBear Software Inc. (siehe auch [URL: LoapUI])
[URL: StarUML] http://staruml.sourceforge.net StarUML - The Open Source UML/MDA Platform
[URL: TestLink] http://sourceforge.net/projects/testlink/ Open Source Tool Test & requirements management
[URL: TestNG] http://testng.org/doc/index.html TestNG, Testframework zum Erstellen von Unit-Tests
[URL: TestToolReview] https://www.testtoolreview.de Informations-Portal Softwaretest-Werkzeuge, imbus AG, Möhrendorf
[URL: TestToolsJava] http://java-source.net/ Open Source Software in Java
[URL: TestToolsOPENSOURCE] http://www.opensourcetesting.org/ Open source tools for software testing professionals
[URL: TestToolsSQA] http://www.softwareqatest.com/qatweb1.html Software QA/Test Resource Center, Rick Hower
[URL: Tidy] http://tidy.sourceforge.net/ HTML Tidy Library Project, SourceForge.net
[URL: UML-OMG] http://www.uml.org/ UML Resource Page, Objekt Management Group
[URL: UserTrackingTools] http://www.sitepoint.com/10-web-analytics-packages-for-tracking-your-visitors/ Liste von User Tracking Tools, SitePoint Pty. Ltd.
[URL: V-Modell] http://www.v-modell-xt.de/ Aktuelles V-Modell XT der Koordinierungs- und Beratungsstelle der Bundesregierung für Informationstechnik in der Bundesverwaltung im Bundesministerium des Innern (KBSt)
[URL: virtualbox] https://www.virtualbox.org/VM VirtualBox®, virtuelle Maschine von ORACLE®
[URL: vmware] http://www.vmware.com/de VMware-Player, virtuelle Maschine von vmware®
[URL: W3C] http://www.w3.org World Wide Web Consortium
[URL: W3C-checklink] http://validator.w3.org/checklink W3C Link Checker
[URL: W3C-Unicorn] http://validator.w3.org/unicorn/ Unicorn - Der Einheitsvalidator des W3C
[URL: W3C-validatorCSS] http://jigsaw.w3.org/css-validator/ W3C CSS-Validierungsservice
[URL: W3C-validatorMarkup] http://validator.w3.org/ W3C Markup Validation Service
[URL: W3C-validatorMobileOK] http://validator.w3.org/mobile/ W3C mobileOK Checker des W3C®
[URL: W3C-WAI] http://www.w3.org/WAI/Web Accessibility Initiative (WAI)
[URL: WAI-Tools] http://www.w3.org/WAI/ER/tools/complete Complete List of Web Accessibility Evaluation Tools, Web Accessibility Initiative (WAI) des W3C
[URL: WAVE] http://wave.webaim.org/ web accessibility evaluation tool, von WebAIM, Utah State University
[URL: WCAG-de] http://www.w3.org/Translations/WCAG20-de/ Autorisierte Übersetzung der Web Content Accessibility Guidelines (WCAG) 2.0 der Deutsche Behindertenhilfe Aktion Mensch e. V., Bonn
[URL: WCAG-ch] http://www.access-for-all.ch/ Autorengruppe Accessibility Checkliste 2.0, Schweiz
[URL: WCAG-ch-CL] http://www.accessibility-checklist.ch/ Checklisten zur WCAG 2.0, Autorengruppe Accessibility Checkliste 2.0, Schweiz

[URL: WCAG-ch-PDF] http://www.access-for-all.ch/en/pdf-lab/pdf-accessibility-checker-pac.html PDF-Accessibility-Checker (PAC 2), Autorengruppe Accessibility Checkliste 2.0, Schweiz
[URL: Webmasterpro] http://www.webmasterpro.de/portal/webanalyse.html Webmasterpro Webanalyse-Reports von Team23 GmbH & Co. KG, Augsburg
[URL: WebDev] https://addons.mozilla.org/de/firefox/addon/web-developer/, Web Developer Toolbar, Firefox Add-on
[URL: wiki-Blick] http://de.wikipedia.org/wiki/Blickbewegung Wikipedia, Stichwort „Blickbewegung“
[URL: XenuLink] http://home.snafu.de/tilman/xenulink.html Xenu's Link Sleuth ™, Link Checker von Tilman Hausherr
[URL: Yahoo!-Regeln] http://developer.yahoo.com/performance/rules.html Best Practices for Speeding Up Your Web Site, Yahoo! Inc.
[URL: YSlow] http://yslow.org/ Tool zur Analyse der Performanzeigenschaften von Webseiten nach Regeln von Yahoo!

Sachverzeichnis

K. Franz, *Handbuch zum Testen von Web- und Mobile-Apps,* Xpert.press,
DOI 10.1007/978-3-662-44028-5