HANSER

Leseprobe

Harry M. Sneed, Manfred Baumgartner, Richard Seidl

Der Systemtest

Von den Anforderungen zum Qualitätsnachweis

ISBN: 978-3-446-42692-4

Weitere Informationen oder Bestellungen unter

http://www.hanser.de/978-3-446-42692-4

sowie im Buchhandel.

4 Systemtestplanung

4.1 Zweck der Testplanung

Der System- bzw. Abnahmetest ist ein Projekt für sich, das neben dem eigentlichen Entwicklungs- bzw. Integrations-, Migrations- oder Installationsprojekt läuft. Als solches muss es wie andere Projekte definiert, kalkuliert, organisiert und geplant werden. Ziel des Testprojekts ist es, die Produktionsreife der vom Hauptprojekt gelieferten Software festzustellen. In der Praxis hat sich ein anderes Ziel manifestiert, nämlich, die Fehler für die Entwickler zu finden. Angesichts des gewohnten Zeitdrucks stellte sich heraus, dass man Zeit gewinnt, wenn der Entwickler von der Last des Testens befreit ist. Außerdem ist so mancher Entwickler angesichts der zunehmenden Komplexität moderner IT-Systeme schlichtweg überfordert. Trotz aller Mahnungen, die eigene Qualität zu sichern, trotz aller neuen Entwicklungsmethoden und ausgeklügelter Modultestwerkzeuge bleibt die Fehlerzahl erschreckend hoch. Die Erfahrung zeigt, dass Software, die vom Entwickler kommt, 3 bis 18 Fehler pro 1000 Anweisungen aufweist [GRAV00]. Mit immer neuen Programmiertechnologien steigt die Fehlerrate. Demnach haben alle Bemühungen, echte Qualität in die Entwicklung zu bringen, nicht gefruchtet. Ob prozedural, objektorientiert oder agentenorientiert, die Entwicklung von Software bleibt ein fehlerhaftes Unterfangen. Und trotz aller Versuche, Entwickler dahin zu bringen, ihre eigenen Komponenten systematisch zu testen, testen sie nach wie vor höchst ungern. Oft fehlt ihnen auch die Zeit dazu. Es gibt psychologische Barrieren, die verhindern, dass Entwickler ihre eigenen Fehler finden wollen [WEIN72]. Ergo bleibt nichts anderes übrig, als jemand anders einzusetzen, der stellvertretend für den Entwickler testet: den Tester, dessen Projekt der System- bzw. der Abnahmetest ist.

Dieses Projekt, der unabhängige Test, erfordert ein eigenes Budget und einen eigenen Zeitplan. Um beides zu beantragen – das Geld wie auch die Zeit –, muss die zu leistende Testarbeit kalkuliert werden. Aus der Testkalkulation geht hervor, wie viele Tester wie lange testen müssen, um alle erforderlichen Testfälle durchführen zu können und notfalls zu wiederholen. Oder, umgekehrt, kommt dabei heraus, wie viele Testfälle die Tester in der vorgegebenen Zeit durchführen können. Ist einmal erkannt, wie viele Tester für wie lange benötigt werden, kann das Testmanagement dazu übergehen, die Testarbeit zu verteilen. Dies ist eine Frage der Testorganisation. Schließlich muss das Testmanagement gewisse Kontrollmechanismen installieren, um den Testfortschritt zu verfolgen, die Testkosten zu überwachen und die Qualität der Testarbeit zu sichern. Darin unterscheidet sich ein Testprojekt nicht von anderen Projekten. Die Voraussetzung dafür ist ein Testplan, in dem die Ziele abgesteckt sind.

Der Testplan regelt, was, warum, wann, wo, wie, womit und von wem getestet wird (siehe Abbildung 4.1). Für die Planung eines strukturierten Systemtests müssen die folgenden Fragen beantwortet werden:

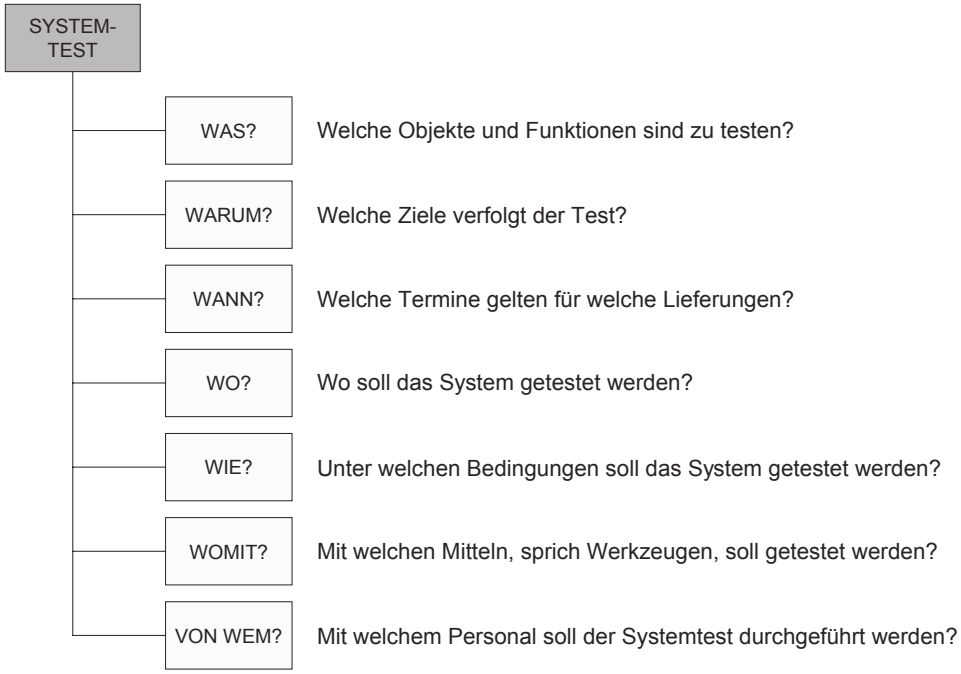

ABBILDUNG 4.1 Die 7 Ws der Testplanung

Das *Was* bezieht sich auf die zu testenden Objekte und Funktionen des Systems. Ein System umfasst verschiedene von außen erkennbare Objekttypen, darunter die Benutzeroberflächen, die Datenbanken, die Berichte und die Systemschnittstellen. Sie gehen aus den Anforderungen hervor, auch dann, wenn sie nicht explizit beschrieben sind und stellen die Gegenstände eines Black-Box-Tests dar. Die Funktionen sind die Anwendungsfälle, die explizit oder implizit in der Anforderungsspezifikation beschrieben sind. Aus der Anforderungsanalyse geht hervor, welche Objekte, Funktionen und Qualitätseigenschaften es zu testen gilt, also die Quantität und Qualität des Produktes. Die Quantität drückt sich in der Zahl der erforderlichen Testfälle aus, die Qualität in den Maßstäben der einzelnen Qualitätsziele. Der Testplan hält den Leistungsumfang des Testprojekts fest. Er bestimmt, wie viele Fälle zu testen und welche Qualitätseigenschaften zu messen sind. Mit dem Testplan verpflichtet sich das Testmanagement, den definierten und vereinbarten Umfang abzudecken (siehe Abbildung 4.2).

Das *Warum* ist eine Frage der Zielsetzung. Was sollte der Test erreichen? Welche Ziele werden angestrebt? Gerade beim Test, einer Arbeit, die sehr undurchsichtig ist, kommt es darauf an, die Leistungen präzise zu definieren. Nach der Grundlektüre „Basiswissen Softwaretest“ hat der Test zwei Hauptziele:

- Fehler zu finden
- Vertrauen in das System zu gewinnen [SPILL02]

Diese Hauptziele können jedoch nur über Nebenziele erreicht werden. Da man nie genau wissen kann, wie viele Fehler eine Software hat, kann man sie nur aufgrund der Erfahrung der Vergangenheit projektieren. Es wird anhand der Fehlerrate der Vergangenheit hochgerechnet. Umso wichtiger ist es, die Fehlerrate aller vergangenen Projekte festzuhalten. Die Fehlerrate ist

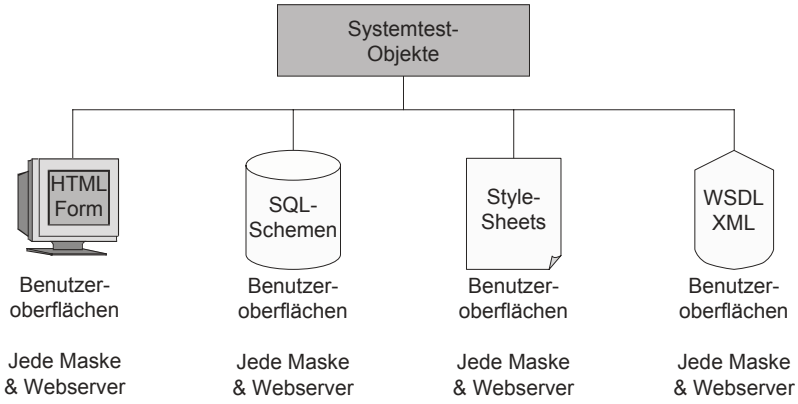

ABBILDUNG 4.2 Beispiele für Systemtestobjekte

das Verhältnis der gefundenen Fehler zur Größe des Projekts, sei es in Anweisungen, Function-Points, Object-Points oder in beliebigen anderen Kategorien. Diese bisherige Fehlerrate wird herangezogen, um die Fehlerrate des geplanten Projekts zu schätzen. Das Vertrauen in der Software wird indirekt über diverse Überdeckungsmaße aufgebaut. Bei der Testplanung geht es darum, diese Überdeckungsgrade mit der Projektleitung abzustimmen und festzuschreiben. Typische Überdeckungsgrade beim Systemtest sind die Überdeckung aller Anforderungen aus dem Fachkonzept, die Überdeckung aller Qualitätsziele und die Überdeckung aller Schnittstellen. Es ist auch möglich, den Softwarecode zu instrumentieren und die Überdeckung aller Source-Bausteine wie Methoden und Prozeduren zu messen. Jedenfalls ist es unerlässlich, eindeutig messbare Ziele – so genannte Testendekriterien – für den Systemtest zu setzen.

Das *Wann* ist eine Frage der Termine. Bis zu welchem Termin wird welcher Test abgeschlossen. Jede Testaktivität soll einen Starttermin und einen Endtermin haben. Welche Zeitpunkte das sind, hängt vom Zeitpunkt der Softwarelieferung und dem Status der anderen Projekt- und Testaktivitäten ab. Es gilt außerdem, gewisse Abhängigkeiten zwischen den Testaktivitäten zu betrachten. Zum Beispiel ist es nicht sinnvoll, mit dem Lasttest zu beginnen, bevor der Funktionstest nicht in einer fortgeschrittenen Phase ist. Die Software wird meistens in Paketen geliefert. Der Test von einem Paket kann erst beginnen, wenn das Paket installiert ist. Oft lässt sich ein Paket nicht testen, bevor nicht ein anderes getestet wurde. Im Testplan werden diese zeitlichen Abhängigkeiten geregelt und die Termine bestimmt, zum Beispiel in Form von Netzplänen oder von Gantt-Diagrammen (siehe Abbildung 4.3).

Das *Wo* ist eine Frage des Orts, an dem der Test ausgeführt wird. Dies muss nicht unbedingt in einem extra dafür bereitgestellten Testraum sein. Die Tester können bei den Entwicklern sitzen oder bei den Endanwendern. Sie können auch zu Hause sitzen oder irgendwo in einem anderen Land. Moderne Kommunikationsmittel ermöglichen es, eine Arbeit von verschiedenen Orten aus gleichzeitig durchzuführen, was jedoch voraussetzt, dass die Arbeit genau spezifiziert ist und alle offenen Fragen per E-Mail geklärt werden. Ist dies nicht der Fall, sollten die Tester zumindest in der Nähe der Entwickler sitzen, um jederzeit persönlich nachfragen zu können. Die Tester müssen auch nicht alle zusammen in einem Raum oder Gebäude sitzen. Wichtig ist nur, dass sie – wo auch immer sie sich aufhalten – Zugriff auf das Testobjekt, die Testwerkzeuge und die Verfasser der Anforderungen haben. Die Frage der Örtlichkeit des Tests sollte im Testplan geregelt werden.

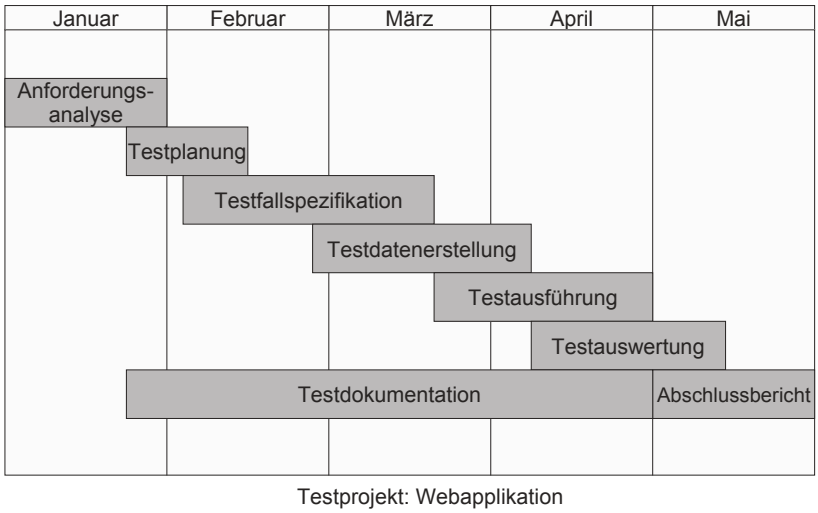

ABBILDUNG 4.3 Testprojektplan

Das *Wie* ist eine Frage der Testmethode. Wie werden die einzelnen Tests durchgeführt? Einen Funktionstest können Tester vom Bildschirmarbeitsplatz aus durchführen, indem sie eingeben, was ihnen gerade einfällt – so etwas nennt man kreatives Testen. Deshalb wurde Testen ursprünglich als Kunst angesehen [MYER79]. Viele Entwicklungsbetriebe versuchten, billiges, unqualifiziertes Personal als Tester einzusetzen. Sie hätten sich genauso gut Affen aus dem lokalen Tiergarten ausleihen und sie auf die Bildschirmarbeitsplätze loslassen können. Deshalb bezeichnet man diesen Test als Affentest [BEIZ95]. Die Ergebnisse blieben unter dem Strich negativ – wenig Geld für wenig Fehler. Die solchermaßen aufgedeckten Fehler sind auch eher trivial. Andererseits kann der Betrieb qualifizierte Tester anweisen, sich strikt an die spezifizierten Testfälle zu halten. Sie sollten die spezifizierten Testfälle vor sich haben und einen nach dem anderen bearbeiten. Dies ist der systematische Test. Er deckt mehr und anspruchsvollere Fehler auf, kostet aber entsprechend mehr. Schließlich könnten die spezifizierten Testfälle dazu benutzt werden, einen Automaten zu bedienen, der sie der Reihe nach durchführt. Das ist der automatisierte Test. Der automatisierte Test bringt die gleichen Vorteile wie der systematische Test, kostet aber nur einen Bruchteil davon, insbesondere im Regressionstest. Der Last- und Performanztest wird ohnehin mit Automaten durchgeführt, weil es gar nicht anders geht. Kurzum: Hier wird entschieden und beschrieben, wie die einzelnen Tests durchzuführen sind. Am Ende muss ein Testprozess festgelegt werden, indem man die einzelnen Testaktivitäten auf die Reihe bringt (siehe Abbildung 4.4).

Das *Womit* ist eine Frage der Testwerkzeuge. Um den Test zu beschleunigen und gleichzeitig die Qualität und Effektivität des Tests zu steigern, werden Werkzeuge eingesetzt. Es beginnt schon mit der Analyse der Anforderungen und der Ermittlung der Testfälle. Auch dafür lassen sich Werkzeuge einsetzen. Für die Validierung der Ergebnisse bzw. den Abgleich der Soll- und Ist-Ergebnisse bieten sich andere Werkzeuge an. Werkzeuge werden auch für das Aufzeichnen und Zurückspielen der Testfälle eingesetzt. Statt manuell zu testen, kann man mit Automaten bzw. Testrobotern testen. Mit Werkzeugen werden Testdaten generiert, Testergebnisse kontrolliert, Testabläufe verfolgt und Testüberdeckung gemessen. Ohne Werkzeuge ist es gar nicht

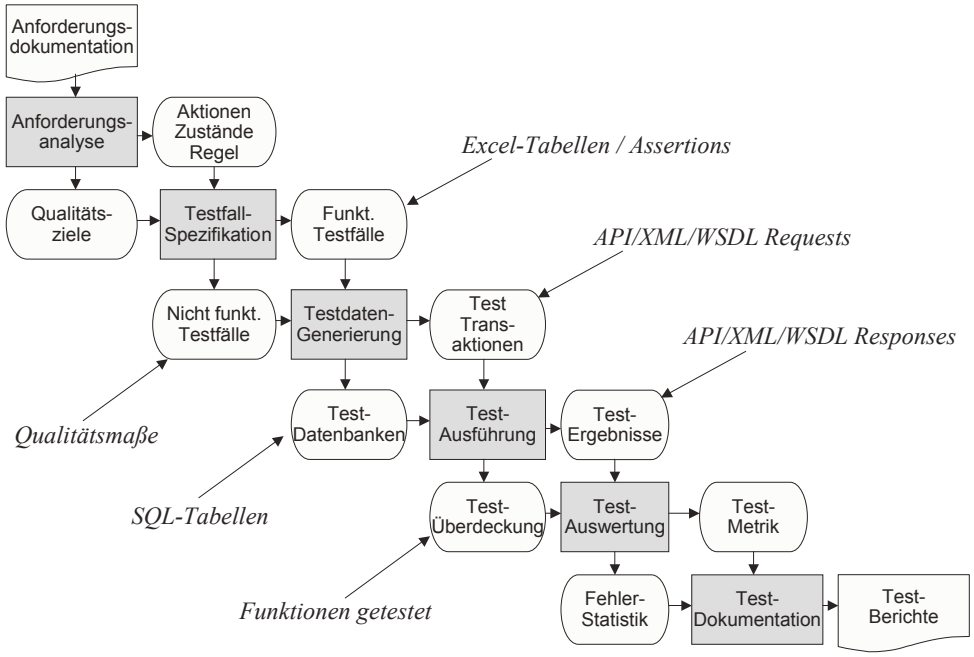

ABBILDUNG 4.4 Festlegung des Testprozesses

möglich, Performanz und Systemauslastung zu messen. Das Fehlermanagement ist ebenfalls nur mit Hilfe eines Werkzeuges möglich. Dennoch wird vor einer übertriebenen Abhängigkeit von Werkzeugen gewarnt. Der Tester muss seine Werkzeuge beherrschen und notfalls für sie einspringen, da er nicht für jede Testaufgabe das geeignete Werkzeug finden wird. Auch wenn das Werkzeug geeignet ist, setzt es eine anspruchsvolle Bedienung voraus. Je komplexer die Aufgabe, desto komplexer das Werkzeug und umso anspruchsvoller die Bedienung. Testautomaten befinden sich immer noch in einer frühen Phase. Deshalb wird das Testmanagement des Öfteren gezwungen, bei vielen Testaufgaben auf qualifizierte Menschen zurückzugreifen. Auf Affen kann es verzichten.

Dies leitet über zur letzten Frage, die der Testplan zu beantworten hat. *Wer* führt die Testaufgaben durch? Auch wenn Testwerkzeuge eingesetzt werden, braucht man Menschen, um die Werkzeuge zu bedienen. Hier geht es also um Arbeitsteilung. Wer macht was? Das Testmanagement muss die zu leistenden Testaufgaben auf das vorhandene Testpersonal verteilen. Dabei soll es auch auf die Neigungen und die Qualitäten der einzelnen Tester Rücksicht nehmen. Dies geschieht in Form einer Zuordnungsmatrix mit zwei Vektoren. Im einem stehen die Aufgaben, im anderen die Namen damit jeweils befassten Mitarbeiter [CRAI02]. So hat das Testmanagement für jede Aufgabe einen Verantwortlichen. Dies entspricht der üblichen Projektorganisation. Allerdings ist sie hier auf das Testprojekt beschränkt (siehe Abbildung 4.5).

Zusammenfassend besteht der Zweck der Testplanung darin, die oben gestellten Fragen – was, wann, wo, wie, womit und von wem – zu beantworten und verbindlich zu dokumentieren. Man sollte genau wissen, wozu man testet. Der Testplan kann als Basis für einen Testauftrag – z. B. einen extern beauftragten unabhängigen Test – dienen.

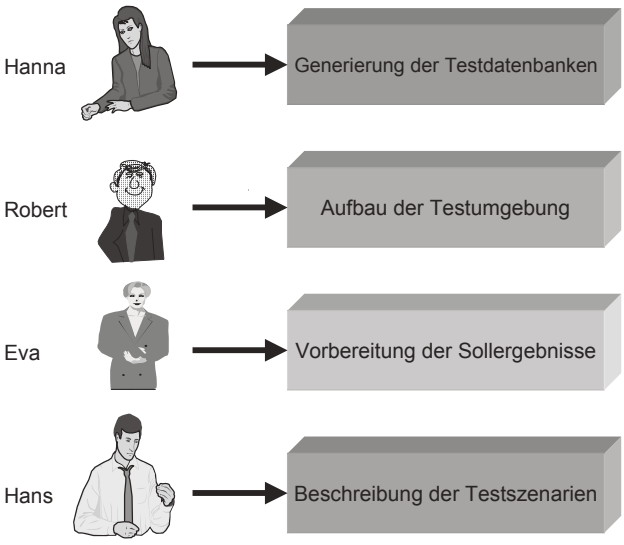

ABBILDUNG 4.5
Zuteilung der Testaufgaben

Ein gutes Beispiel für den Vergleich eines personalintensiven Massentests mit einem schlanken, automatisierten Systemtest liefert ein Projekt eines der Autoren dieses Buches. So arbeiteten 2003 noch ca. 90 Tester in der Systemtestabteilung, meistens Studenten und Hilfskräfte. Nur das Führungspersonal war fest angestellt. Es waren Releases mit rund 2,5 Millionen Anweisungen vierteljährlich zu testen, hinzu kamen zwischenzeitlich Hotfixes und Patches. Von 10 Fehlern wurden 6 vor der Auslieferung gefunden. Heute arbeiten ca. 25 hoch qualifizierte, fest angestellte Tester in der gleichen Testabteilung in einem hoch automatisierten Testprozess und finden in der gleichen Zeitspanne 9 von 10 Fehlern. Es lohnt sich also durchaus, gut ausgerüstete, qualifizierte Tester einzusetzen [KUFN11].

4.2 Voraussetzungen zur Testplanung

Eine systematische Testplanung, die die oben gestellten Fragen sinnvoll beantwortet, setzt gewisse Informationen voraus, ohne die eine genaue Planung nicht möglich ist. Wer planen will, was zu testen ist, muss wissen, welche und wie viele Fälle zu testen sind. Diese Informationen lassen sich bei neuen Systemen nur aus den Anforderungen und bei bestehenden Systemen nur aus der Systemdokumentation bzw. der Codeanalyse ableiten. Darum ist die Anforderungsanalyse eine unabdingbare Voraussetzung für die Testplanung. Sie muss vor der Testplanung stattfinden. Beim Komponententest ist es erforderlich, erst den Source-Code der Komponente zu analysieren, um die Ziele des Tests abzustecken. Beim Integrationstest ist es erforderlich, erst die Architektur des Systems zu analysieren. Beim Systemtest ist es das Konzept bzw. die Anforderungsspezifikation, welche vorher zu analysieren ist, um den Umfang des Systemtests abzustecken.

Eine Herausforderung ist, dass für die Testplanung erst die Testbasis analysiert werden muss. Beim Systemtest ist die Testbasis entweder das Anforderungsdokument oder das Systemmodell. Das ist auch der Grund, weshalb wir in diesem Buch die Anforderungsanalyse und die Systemmodellierung vor die Testplanung stellen. Bevor die Testplanung begonnen wird, muss die Analyse der Testbasis abgeschlossen sein. Das bedeutet wiederum, dass eine Sacharbeit vor der Planungsarbeit stattfinden muss. Dies widerspricht zwar dem Grundsatz „erst planen, dann arbeiten", ist aber erforderlich, um nicht umsonst zu planen.

In der Praxis kommt es sehr häufig vor, dass in Projekten geplant wird ohne zu wissen, was zu planen ist. Dieser Umstand ist einer der Gründe für die Entstehung der agilen Entwicklungs- und Testmethoden. In den agilen Vorgehensmodellen werden Planung und Ausführung in kurzen Iterationen ausgeführt [LINK09]. Ein agiles Projekt ist per Definition ein nicht-deterministischer Prozess. Der Umfang und Inhalt kann sich während der Entwicklung noch ändern, und keiner kann am Anfang sagen, wo die Entwicklung endet, geschweige denn, wie lange sie dauert und wie viel sie kostet [BLOC11].

Klassische Projekte funktionieren aber auch sehr gut, wenn vorher gründlich genug analysiert wird. Wenn die Anforderungen vorher wirklich im Detail ausgearbeitet und eventuell sogar modelliert werden, wird die Information für eine systematische Planung ausreichen.

Zur Definition der Testziele – des Warum – werden zum einen die Größenmaße des Systems – zum Beispiel die Größe in Function-Points, Object-Points, Test-Points oder Anweisungen – und zum anderen die Fehlerrate vergangener oder ähnlicher Projekte benötigt. Für neue Systeme müssen die Größenmaße aus der Analyse der Anforderungen gewonnen werden. Für existierende Systeme, die ersetzt, saniert, migriert oder integriert werden sollten, ist es notwendig, das alte System erst durch Reverse Engineering nachzudokumentieren und anschließend die Größenmaße aus der Nachdokumentation zu gewinnen. Die Fehlerrate bisheriger Projekte lässt sich aus einer Analyse der Fehlerdatenbank ermitteln. Ohne diese Messwerte wird es kaum möglich sein, messbare Testziele zu setzen.

Die Bestimmung der Testtermine – des Wann – hängt von den Lieferterminen ab. Diese werden im allgemeinen Testplan festgelegt. Demzufolge setzt die Testplanung die allgemeine Projektplanung voraus. Als Erstes muss ein Projektplan vorliegen, dann kann der Testplan darauf aufbauen.

Die Regelung der Testörtlichkeit – des Wo – setzt Kenntnisse der Verfügbarkeit voraus. Der Testmanager muss zum Zeitpunkt der Testplanung bereits wissen, mit welcher Hardware-Ausstattung getestet wird und ab wann diese Geräte verfügbar sind. Die Entscheidung, den Test zu verteilen oder zentralisiert durchzuführen, hängt u. a. auch von der Verfügbarkeit der Rechnerressourcen ab.

Die Wahl der richtigen Testmethode – des Wie – setzt voraus, dass der Testplaner die Art der Anwendung kennt. Er muss wissen, worauf es ankommt, wo die Risiken liegen und wo die meisten Fehler auftauchen könnten. Nur mit diesen Kenntnissen ist er in der Lage, die geeignete Vorgehensweise auszuwählen und diese wiederum dem Projekt anzupassen. Hier werden also sowohl System- als auch Anwendungskenntnisse gefordert. Darüber hinaus muss der Testplaner mit den gängigen Systemtestmethoden vertraut sein. Sonst wird er nicht wissen, welche Ansätze überhaupt zur Auswahl stehen.

Zum Einsatz von Testwerkzeugen – dem Womit – ist es erforderlich zu wissen, welche Werkzeuge zur Auswahl stehen und für welche Aufgaben die Werkzeuge geeignet sind. Denn im Gegensatz zum Entwurf eines neuen Systems, bei dem meistens ein Werkzeug ausreicht,

braucht man beim Test eines Systems viele verschiedene Werkzeuge – für jede Art von Testaktivitäten eines. Der Testplaner benötigt Informationen zur Funktionalität und Qualität der in Frage kommenden Werkzeuge. Er muss auch wissen, wie viel Zeit die Einarbeitung in die Handhabung der Werkzeuge beansprucht. Wenn der Termin zu knapp ist, werden die Tester nicht die Zeit haben, sich mit den Werkzeugen vertraut zu machen. Andererseits werden manche Aufgaben ohne Werkzeug nicht zu bewältigen sein. Der Testmanager steht hier vor einer sehr schweren Entscheidung.

Zur Verteilung der Testarbeit – dem Wem – muss der Testmanager seine Mitarbeiter kennen und wissen, wozu sie imstande sind. Auf der einen Seite stehen die zu bewältigenden Testaufgaben, auf der anderen Seite die verfügbaren Mitarbeiter. Man muss beide kennen, um sie einander zuordnen zu können. Auch das Testen verlangt spezielle Kenntnisse. Seinen Skills entsprechend werden jedem Tester andere Themen vertraut sein, z. B. auf den Gebieten der Technik, Methodik oder Fachlichkeit. Es obliegt dem Testmanagement, die Tester optimal zu koordinieren.

Schließlich benötigt der Testplaner einige Daten aus der Vergangenheit, um die Testaufwände und die Testdauer abschätzen zu können. Zum einen geht es um die bisherige Fehlerrate. Zur Berechnung der Anzahl der zu erwartenden Fehler braucht der Planer Angaben über die Anzahl gefundener Fehler in den bisherigen Projekten. Die Fehlerdichte, das heißt die Anzahl der Fehler pro tausend oder hundert Größeneinheiten, ist ein Maß, das sich auf das neue System übertragen lässt. Die Größeneinheit könnten Codezeilen, Anweisungen, Function-Points oder auch Testfälle sein. Dieses Maß muss durch den Grad der Testüberdeckung relativiert werden. Deshalb genügt es nicht, nur die absolute Anzahl der Fehler zu kennen, man muss auch die Größe der betroffenen Software und den Grad der Testüberdeckung kennen, wie wir bei der Schätzung der Testaufwände gleich sehen werden.

Zum anderen geht es um die Produktivität der Tester. Tester sollten Fehler finden und Vertrauen in das System aufbauen. Dazu müssen sie möglichst viele und möglichst effektive Testfälle ermitteln. Die dafür benötigte Zeit hängt von ihrer Produktivität ab. In der Softwareentwicklung ist die Produktivität ein Maß für die Anzahl der Größeneinheiten, die ein Entwickler pro Zeiteinheit erstellen kann, zum Beispiel die Anzahl der Anweisungen pro Tag oder Function-Points pro Monat. Beim Systemtest ist die Produktivität ein Maß für die Anzahl der Testfälle, die ein Tester pro Zeiteinheit auszuführen in der Lage ist. Natürlich wird dies von Projekt zu Projekt variieren und hängt u. a. vom Grad der Testautomatisierung ab. Dennoch lassen sich künftige Aufwände nur aufgrund der bisherigen Produktivität voraussagen. Der Testplaner benötigt eine Erfahrungsdatenbank, aus der das Verhältnis geleisteter Testertage zu ausgeführten Testfällen hervorgeht.

Diese beiden Metriken – die Fehlerrate und die Testproduktivität – sind unerlässliche Voraussetzungen für den nächsten Schritt – die Schätzung der Testaufwände.

4.3 Schätzung der Testaufwände

Ein Plan lässt sich nicht erstellen, wenn man nicht weiß, wie viel Kapazität und wie viel Zeit eine Aufgabe erfordert. Auch wenn die Projektumstände Zeit und Kapazität diktieren, muss der Testmanager über Zeit und Aufwand des Tests Bescheid wissen. Jede Aufgabe, auch der Systemtest, braucht ihre Zeit. Zeit ist wiederum eine Frage des Aufwands und lässt sich durch den Einsatz von mehr Personal verkürzen, was aber nur bedingt möglich ist. Das Verhältnis zwischen Aufwand und Zeit ist nicht linear, das heißt, wir können ein Projekt von 12 Mannmonaten nicht auf 6 Mannmonate reduzieren, indem wir die Anzahl der Projektbeteiligten verdoppeln. Diese Annahme ist ein Softwaremythos [BROK75]. Je mehr Menschen an einem Projekt arbeiten, desto größer ist der Kommunikations-, Organisations- und Administrationsaufwand. Die Produktivität sinkt mit jedem zusätzlichen Mitarbeiter. Zeit und Aufwand sind daher nicht beliebig austauschbar, wie alle Studien zur Softwareaufwandskalkulation beweisen. Sie beeinflussen sich gegenseitig. Der Schlüssel zur Ermittlung beider ist die Produktivität (siehe Abbildung 4.6).

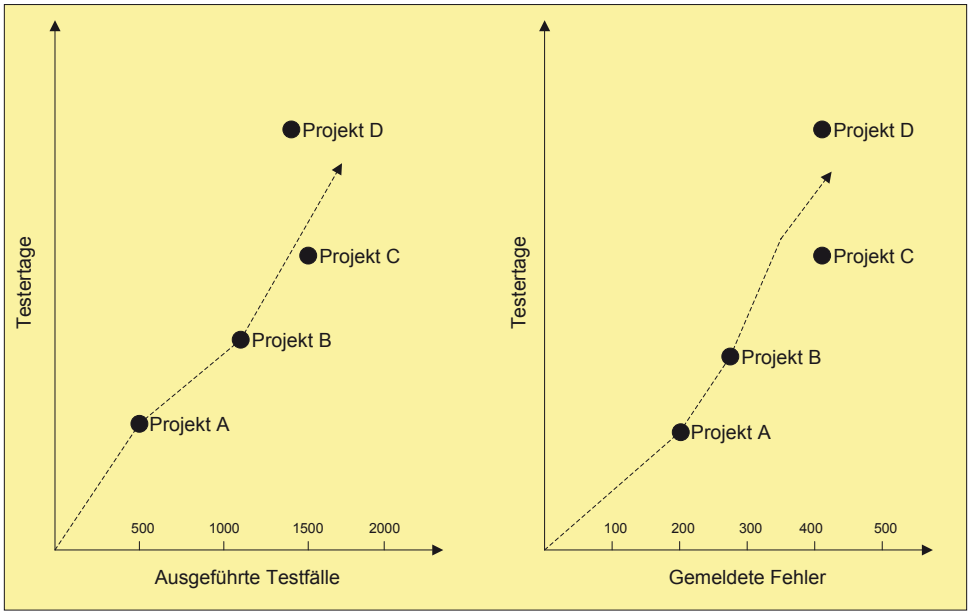

ABBILDUNG 4.6 Ermittlung der Testproduktivität

4.3.1 Test-Points

Testproduktivität drückt sich in der Anzahl der Test-Points pro Testertag aus. Test-Points sind die Anzahl der Testfälle, ergänzt durch die Anzahl der zu testenden System- und Benutzerschnittstellen. Da jede Schnittstelle einen zusätzlichen Aufwand bedeutet, muss sie in die Kalkulation einbezogen werden. Der Begriff Test-Points stammt ursprünglich von einem belgischen Testanbieter, der sein Testaufwandsschätzverfahren auf einer STAR-Konferenz im Jahre 1999 vorstellte [COLL99]. Es stellt den Versuch dar, etwas Ähnliches wie die Function-Points für die Entwicklung auch für den Test einzuführen. Dies ist insofern gelungen, als die holländische Firma Sogeti die Methodik in ihr Testverfahren TMAP aufnahm und weltweit propagierte [KABV06]. Nach TMAP ist die endgültige Anzahl der Test-Points die Summe der dynamischen und der statischen Test-Points. Die dynamischen Test-Points sind äquivalent zur Anzahl der logischen Testfälle aus den Anforderungen. Die Anzahl statischer Test-Points wird aus der Anzahl der Testobjekte abgeleitet. Als Testobjekte sind im Systemtest die zu testenden Systemschnittstellen, Benutzeroberflächen und Datenbanken anzusehen. Diese sind übrigens auch die Objekte der Function-Point-Analyse. Martin Poll hat für TMAP den Begriff TOSM – Test Object Size Meter – geprägt, wonach ein Test-Point in etwa 1,5 bis 3 Arbeitsstunden entspricht [KOPO99].

Jedes Testobjekt sollte wie bei der Function-Point-Methode in Abhängigkeit von seiner Komplexität gewichtet werden. Da zum Zeitpunkt der Schätzung die Komplexität der einzelnen Testobjekte aber selten bekannt ist, empfiehlt es sich, den Objekten einen Standardwert zuzuweisen. Der im letzten Kapitel vorgestellte Textanalysator zählt nicht nur die logischen Testfälle bzw. die dynamischen Test-Points, sondern auch die Anzahl der Testobjekte, zumindest die Benutzeroberflächen, die Berichte und die Systemschnittstellen, um die statischen Test-Points wie folgt zu errechnen.

$$\text{TestPoints} = \text{Nr\_TestCases} + (\text{Nr\_Panels} \times 4) + (\text{Nr\_Reports} \times 2) + (\text{Nr\_Services} \times 1)$$

4.3.2 Testproduktivität

Die Testproduktivität hängt im Wesentlichen vom Grad der Testautomatisierung und der Erfahrung und Motivation der Tester ab. Leider lässt sie sich nicht einfach „aus der Luft holen“. Produktivität will gemessen werden und zwar anhand der Statistik bisheriger Projekte. Also muss der Testmanager wissen, wie viele Test-Points es pro Projekt gegeben hat und wie viele Testertage dafür gearbeitet wurde. Dazu muss er die Testumstände bzw. die Einflussfaktoren berücksichtigen, um diese Beziehung bewerten zu können [ANSE03]. Demnach ist:

$$\text{Testproduktivität} = \frac{\text{TestPoints}}{\text{Testertage}} \times \text{Einflussfaktor}$$

Die Testproduktivität zu kennen, ist leider nicht genug, um den Aufwand für ein neues Testprojekt zu kalkulieren. Es gehört mehr dazu, denn jedes Produkt hat seine Besonderheiten, und jedes Projekt findet unter anderen Bedingungen statt. Daher empfiehlt sich die COCOMO-II Formel von Boehm für die Schätzung des Testaufwands [BOEH99]. Außer der Produktivität berücksichtigt COCOMO-II drei weitere Faktoren:

- Projektbedingungen
- Produktqualität
- Systemtyp

Die Projektbedingungen werden in einem Exponenten von 0,91 bis 1,23 zusammengefasst. Die fünf Bedingungen, die Boehm vorschlägt und die zu einem Testprojekt relativ gut passen, sind:

- Grad der Testwiederholbarkeit
- Stabilität der Testumgebung
- Kenntnis der Anwendung
- Zusammengehörigkeit des Testteams
- Reife des Testprozesses

Jede dieser Bedingungen wird auf der nominalen Skala

- Sehr niedrig = 1,23
- Niedrig = 1,10
- Mittel gut = 1,00
- Hoch = 0,96
- Sehr hoch = 0,91

eingestuft. Der Skalierungsexponent ist der arithmetische Mittelwert dieser fünf Bedingungen. Wenn also

- die Testwiederholbarkeit mittel gut,
- die Stabilität der Testumgebung hoch ist,
- die Kenntnisse der Anwendung niedrig sind und
- die Zusammengehörigkeit des Testteams niedrig sowie
- die Reife des Testprozesses mittel gut ist,

wäre die Skalierungsexponente:

$$\frac{1{,}0 + 0{,}96 + 1{,}10 + 1{,}10 + 1}{5} = 1{,}03$$

4.3.3 Komplexität und Qualität

Um den Testaufwand zu kalkulieren, müssen zwei weitere Faktoren berücksichtigt werden – Qualität und Komplexität. Die Softwarequalität ist aus Sicht des Tests deren Testbarkeit, das heißt, wie leicht bzw. wie schwer es ist, dieses System zu testen. Zu messen ist dies anhand des Aufwandes, der erforderlich ist, um ein gegebenes Testziel zu erreichen. Je höher dieser Aufwand, desto niedriger die Testbarkeit. Breite Schnittstellen mit vielen Parametern und überladene graphische Oberflächen mit vielen Widgets sind Zeichen niedriger Testbarkeit.

Die Softwarekomplexität ist eine Frage der Anzahl der Beziehungen zwischen Entitäten. Hier kommen mindestens vier Komplexitätsmaße in Frage:

- Komplexität der Benutzeroberflächen
- Komplexität der Systemschnittstellen

- Komplexität der Datenbanken
- Komplexität der Anwendungsfälle [KROP03]

Alle vier Komplexitätsmaße sind rationale Maße auf der Skala von 0,0 bis 1,0. Die Komplexität der Benutzeroberfläche ist das Verhältnis der Anzahl der Steuerungseingaben zur Summe der Oberflächeneingaben insgesamt. Je mehr Steuerungseingaben – Radio-Buttons, Menüeinträge, Checkboxen usw. –, desto komplexer der Test einer Benutzeroberfläche, wohingegen einfache Textfelder, die den weiteren Ablauf nicht beeinflussen, die Komplexität nicht erhöhen.

$$\frac{\text{Steuerungseingaben}}{\text{Gesamteingaben}}$$

Die Komplexität der Systemschnittstellen ist das Verhältnis der Anzahl verschiedener Datentypen zur Anzahl der Datenelemente pro Schnittstelle. Je mehr verschiedene Datentypen es gibt, desto aufwändiger ist es, eine Systemschnittstelle zu testen.

$$\frac{\text{Datentypen}}{\text{Datenelemente}}$$

Die Komplexität der Datenbanken ist das Verhältnis der Schlüssel (Primär- und Fremdschlüssel) und der Indizes zur Gesamtanzahl aller Attribute. Je mehr Attribute einer Datenbank Teil von Primär- und Fremdschlüsseln sind und je mehr Attribute indiziert sind, umso schwieriger ist die Testdatenaufbereitung.

$$\frac{\text{Schlüsselattribute} + \text{Indizes}}{\text{Attribute}}$$

Die Komplexität der Anwendungsfälle ist schließlich das Verhältnis der Bedingungen zu den Aktionen. Je mehr Bedingungen einen Vorgang steuern, desto größer ist der Aufwand, den Vorgang bzw. den Anwendungsfall zu testen.

$$\frac{\text{Bedingungen}}{\text{Aktionen}}$$

Möglicherweise wird der Testplaner nicht über genügend detaillierte Systeminformationen verfügen, um diese Zahlen zu ermitteln. In diesem Fall kann er die Komplexität mit Hilfe einer nominalen Skala schätzen:

- Sehr hoch = 0,8
- Hoch = 0,6
- Durchschnittlich = 0,5
- Niedrig = 0,4
- Sehr niedrig = 0,2

Die Systemtestkomplexität ist der arithmetische Mittelwert der einzelnen Testkomplexitäten. Der Testbarkeitsfaktor ist die gemessene oder geschätzte Systemkomplexität, dividiert durch die mittlere Komplexität = 0,5 [SNEE06].

$$\text{Testbarkeitsfaktor} = \frac{\text{gemessene Komplexität}}{\text{mittlere Komplexität}}$$

4.3.4 Die COCOMO-II Gleichung

Der Systemtyp hat eine große Auswirkung auf den Aufwand. Boehm teilt die Systeme in vier Kategorien ein:

- Standalone-Systeme, Single-User-Systeme
- Verteilte Multi-User-Systeme
- Integrierte, verteilte Systeme mit mehreren Benutzern
- Embedded-Systeme

Standalone Systeme sind mit 0, verteilte Systeme mit 1, integrierte Systeme mit 2 und Embedded Systeme mit 4 gewichtet. Wesentlich ist, dass man nach Möglichkeit immer mit dem gleichen Systemtyp vergleicht. Der Systemtyp darf nur in die Berechnung einbezogen werden, wenn das geplante System von einem anderen Typ ist als das System, von dem die Produktivität entnommen wird. Mit diesen Parametern lässt sich der Testaufwand mit folgender Gleichung schätzen:

$$\text{Testaufwand} = \text{Systemtyp} \times \left(\frac{\text{Test-Points}}{\text{Testproduktivität}} \right)^{\text{Skalierungsexponente}} \times \text{Testbarkeitsfaktor}$$

Angenommen, es werden 1600 Test-Points anhand der Konzeptanalyse gezählt. Im letzten ähnlichen Projekt wurden 1000 Test-Points in 150 Personentagen getestet. Das ergibt eine Testproduktivität von 1000/150 = 6,6 Test-Points pro Personentag.

Die Skalierungsexponente für dieses Projekt wird wie folgt eingeschätzt.

- Testwiederholbarkeit = mittel = 1,00
- Stabilität der Testumgebung = hoch = 0,96
- Kenntnis der Anwendung = niedrig = 1,10
- Zusammengehörigkeit des Testteams = hoch = 0,96
- Testprozessreife = hoch = 0,96

Daraus folgt der arithmetische Mittelwert 0,99.

Die Analyse der Software deutet auf eine überdurchschnittliche Komplexität dieses Systems von 0,60 hin. Durch die Komplexität des Vergleichssystems von 0,5 ergibt dies einen Testbarkeitsfaktor von 1,2. Schließlich handelt es sich um denselben Systemtyp wie bisher – ein verteiltes Client/Server-System, also Typ 1. Demzufolge ist der geschätzte Testaufwand:

$$\text{Testaufwand} = 1 \times \left(\frac{1600}{6{,}6} \right)^{0{,}99} \times 1{,}2 = 275 \text{ Personentage}$$

Die geringfügig besseren Projektbedingungen, die durch den Exponent 0,99 erfasst sind, werden durch die 20 % überdurchschnittliche Komplexität des neuen Systems ausgeglichen, so dass am Ende die Testproduktivität bei 5,8 Testpunkten pro Testertag 12 % niedriger ausfällt. Dieser Aufwand umfasst genau jene Testaktivitäten, die die alte Produktivitätsmessung umfasst. Wenn dort der Aufwand für die Testplanung nicht mit erfasst wurde, wird er auch hier nicht berücksichtigt. Darum ist es unerlässlich, bei der Produktivitätsmessung konsistent zu bleiben und immer die gleichen Aufwände zu erfassen. Nur so lässt sich eine genaue Hochrechnung für künftige Projekte sichern [SNEE03].

4.4 Schätzung der Testdauer

Nach dem COCOMO-Modell von Boehm besteht ein algorithmisches Verhältnis zwischen dem Aufwand für ein Projekt und der Projektdauer. Die Gleichung für die Dauer einer Softwareentwicklung war in COCOMO-I:

$$\text{TDEV} = 2{,}5 \times (\text{Aufwand})^{\text{Exponent}}$$

wobei der Exponent folgendermaßen gewählt wird:

- 0,38 für Standalone-Systeme
- 0,35 für verteilte Systeme
- 0,32 für Embedded-Realtime-Systeme [BOEH84]

Nach der ursprünglichen COCOMO-I-Formel wäre die Dauer eines Testprojekts mit dem geschätzten Aufwand von 275 Personentagen bzw. 14 Personenmonaten:

$$\text{TDEV} = 2{,}5 \times (14)^{0{,}35} = 6{,}3 \text{ Monate}$$

Das wäre die minimale Dauer des Projekts mit zwei Testern. COCOMO-I geht jedoch von der Entwicklung als nur bedingt teilbare Aufgabe aus. Testprojekte sind aber wie Migrationsprojekte besser teilbar. Bis auf die Anfangsphasen der Analyse und Planung können Tester besser nebeneinander arbeiten. Damit lässt sich die Dauer eines Testprojekts zwar nicht ganz, aber doch stark komprimieren. Daher ist die COCOMO-II-Zeitformel für den Test besser geeignet.

Im neuen COCOMO-II-Modell ist die Gleichung:

$$\text{TDEV} = (\text{C} \times (\text{PM})^{\text{F}}) \times \left(1 - \frac{\text{SCED \%}}{100}\right)$$

wobei

$$\text{F} = (\text{D} + 0{,}2 \times (\text{E} - \text{B}))$$

In dieser Gleichung sind die Parameter wie folgt definiert:

- B = Untergrenze der Skalierungsexponente = 0,91
- C = Multiplikator = 3,67 für Neuentwicklung
- D = Zeitbasiskoeffizient = 0,28 für Neuentwicklung
- E = Skalierungsexponent der Entwicklung = 0,91 : 1,23
- F = Skalierungsexponent für die Projektdauer
- PM = geschätzter Aufwand in Personenmonaten
- SCED % = % Kompression des Projekts
- TDEV = Zeit für die Entwicklung [BOEH00]

Danach ist die Dauer des Testprojekts wie folgt zu berechnen:

$$\text{F} = (0{,}28 + 0{,}2 \times (0{,}96 - 0{,}91)) = 0{,}29$$

$$\text{TDEV} = 3{,}67 \times (14)^{0{,}29} = 7{,}9 \text{ Monate}$$

Dies ist die Testdauer, wenn man, wie im früheren Projekt, zwei Tester einsetzt. Nun kommt aber der Schedule-Compression-Faktor hinzu, und dieser ist beim Testen hoch. Wir können die Zeit zum Testen durch die Verteilung der Arbeit auf mehrere Tester um mindestens 60 % verkürzen. Das heißt:

$$\text{TDEV} = 7{,}9 \times \left(1 - \frac{60}{100}\right) = 3{,}2 \text{ Monate}$$

Demnach könnten wir z. B. fünf Tester einsetzen. Anforderungsanalyse und Testplanung werden von zwei Testern bewerkstelligt. Danach kämen drei weitere hinzu, die zusammen mit den beiden ersten die Testfälle spezifizieren, die Testdaten generieren, die Testumgebung aufbauen, den Test durchführen und die Testergebnisse auswerten. Das gesamte Testprojekt könnte innerhalb von drei Monaten abgeschlossen sein. Es ist schließlich eine Frage der Projektorganisation und der Rollenverteilung.

4.5 Testprojektorganisation

Testen ist in großem Ausmaß von der Umgebung abhängig. Spezifizieren, Entwerfen und Codieren sind Aktivitäten, die auch ohne Rechnernutzung ausgeführt werden könnten bzw. der Rechner wird nur zum Zeichnen und Texte-Editieren verwendet – Aufgaben, die im Prinzip per Hand zu erledigen wären. Im Grunde benötigt man den Rechner erst beim Kompilieren (wie in der Frühzeit der Softwareentwicklung). Zum Testen war man aber schon immer auf den Rechner angewiesen.

4.5.1 Organisation der Testressourcen

Es gab Zeiten, als Rechenkapazität noch nicht im Überfluss vorhanden war. Damals konnte es vorkommen, dass die Rechnernutzung für Tests zum Engpass des Projekts wurde. Nur eine begrenzte Anzahl von Menschen durfte eine bestimmte Zeit lang mit dem Rechner arbeiten. Falls der Test im Batch-Betrieb lief, waren die Tester gezwungen, ihre Tests als Batchjobs auf Band, Lochkarten oder Lochstreifen vorzubereiten und dann bei den Operatoren abzugeben. Auf diese Weise konnte man pro Tag maximal drei Tests fahren. Notfalls konnte ein Projekt den Rechner für sich bekommen, um mehrere Tests hintereinander zu fahren, doch war dies nur nachts möglich. Bei seinen ersten Testprojekten Ende der 70er-Jahre durfte der Autor nur nachts arbeiten, bei Siemens in München von 16.00–24.00 Uhr und bei der Spardat in Wien von 18.00–02.00 Uhr. Ergo musste das Testen sehr strikt organisiert werden, um die beschränkte Rechnerzeit optimal auszunutzen. Oft musste die Rechnerzeit Monate im Voraus bestellt werden. Heute sieht das anders aus: Rechnerkapazität ist im Überfluss vorhanden, dafür gibt es andere Engpässe.

Der Hauptengpass beim heutigen Testen ist nicht die Rechnernutzung, sondern die Datennutzung. Zum Testen braucht man einen Bildschirmarbeitsplatz, eine Verbindung zum Server, die Software auf dem Server und die Daten auf dem Server. Die Serversoftware ist in der Regel multiserverfähig, und wenn nicht, lässt sie sich für jeden Test duplizieren. Leider trifft dies für die Daten nicht immer zu. Für den Test braucht jeder Tester seine eigenen Daten, um den Vorzustand jederzeit wiederherstellen und somit den Nachzustand validieren zu können, ohne Gefahr zu laufen, dass eine andere Person seine Daten in der Zwischenzeit absichtlich oder unabsichtlich verändert. Es ist deshalb dringend notwendig, die Testdaten für jeden Tester getrennt zu halten. Dies hört sich banal an, ist aber für viele Projekte ein großes Problem, weil die Plattenspeicherkapazität nicht ausreicht oder weil das Datenbanksystem nur eine Kopie einer Datenbank zulässt [MOSL00].

Die Verfügbarkeit der Testdaten ist ein organisatorisches Problem, das es bereits bei der Testplanung zu berücksichtigen gilt. Der Testmanager muss dafür sorgen, dass jeder Tester einen Testarbeitsplatz mit Verbindung zum Server hat, dass der Server genügend Hauptspeicherkapazität für die gleichzeitige Bedienung aller Tester hat, dass das Kommunikationsnetz die Testlast verträgt sowie – vor allem –, dass genügend externe Speicherkapazität vorhanden ist, um jedem Tester eine eigene Testdatenbank zur Verfügung zu stellen. Die Datenbankkommunikation muss auch gewährleisten, dass das Datenbanksystem mehrere Kopien der gleichen Datenbank nebeneinander verwalten kann. Die Ressourcenplanung ist also ein wichtiger Bestandteil der allgemeinen Testplanung und darf nicht vernachlässigt werden. Es kommt darauf an, Kapazitätsengpässe bei der Testdurchführung zu vermeiden. Dies erreicht man am besten, wenn der Kapazitätsbedarf rechtzeitig geregelt wird (siehe Abbildung 4.7).

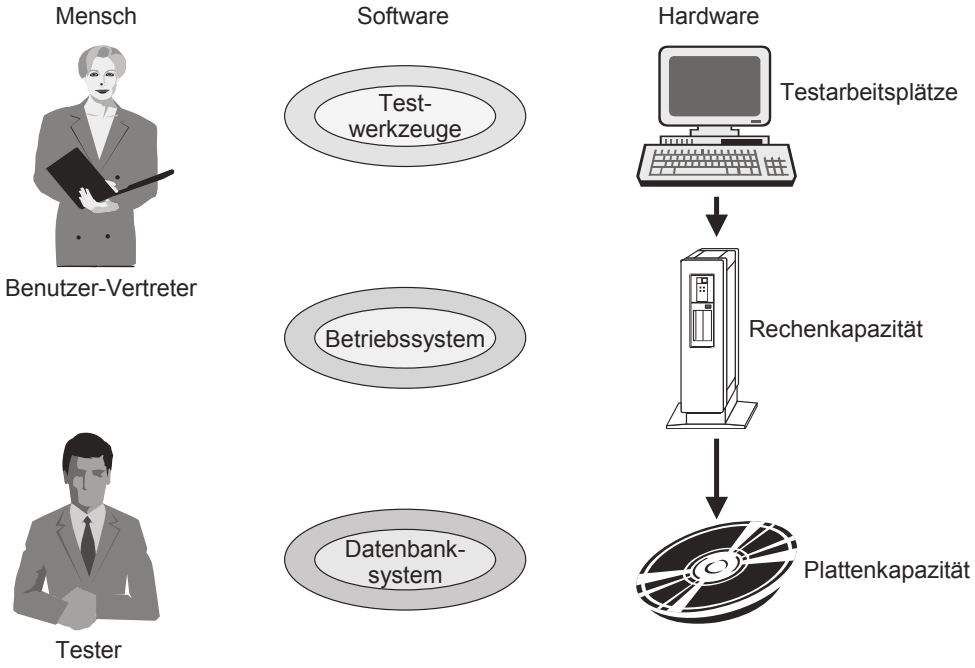

ABBILDUNG 4.7 Einteilung der Testressourcen

4.5.2 Organisation des Testpersonals

Die Bereitstellung der nötigen Maschinenressourcen ist nur die eine Seite der Testorganisation. Die andere Seite ist die Bereitstellung der nötigen Humanressourcen. Menschen sind erforderlich, um die Testfälle zu spezifizieren, die Testdaten zu ermitteln, die Testumgebung aufzubauen, den Test durchzuführen und die Testergebnisse zu kontrollieren. Auch wenn Testwerkzeuge zu diesem Zweck eingesetzt werden, braucht man Menschen, um die Werkzeuge zu bedienen. Hier stellt sich die Frage, welche Menschen gebraucht werden und wie viele. Ein Entwickler eignet sich nicht immer als Tester. Viele Entwickler haben eine Abneigung gegen das Testen und sind dafür nicht zu begeistern. Sie haben weder die nötige Geduld und Ausdauer noch die nötige Aufmerksamkeit für Details. Außerdem fehlen ihnen die Fachkenntnisse. Dafür hätten sie die erforderlichen technischen Kenntnisse und Fähigkeiten. Sachbearbeiter von der Fachabteilung bringen die Fachkenntnisse der Anwendung mit. Sie verfügen auch meist über die nötige Ausdauer und Aufmerksamkeit in Detailfragen. Was ihnen fehlt, sind die erforderlichen technischen Kenntnisse und Fähigkeiten sowie das Vermögen, sich in die Feinheiten der Softwarekonstruktion hineinzudenken. Demzufolge können sie sich nur schwer vorstellen, welche Art von Fehler auftreten könnte. Systemanalytiker bzw. diejenigen, die zwischen den Anwendern und den Entwicklern vermitteln sollten, sind die am besten geeigneten Kandidaten. Sie besitzen die technischen Fähigkeiten und die besten Kenntnisse hinsichtlich des Systems. Immerhin haben sie die Anforderungen spezifiziert, die jetzt getestet werden sollten. Was ihnen fehlt, sind Geduld, Aufmerksamkeit für Details und die nötige Ausdauer. Hinzu kommt, dass sie sich mit Routinetestarbeiten nicht gerne abgeben. Analytiker sind kreative Menschen, die nur ungern wiederholte Tätigkeiten ausführen. Summa summarum: Es ist nicht ganz einfach, geeignete Tester zu finden. Im Grunde genommen braucht man eine Mischung aus Entwickler, Analytiker und Anwender, die noch dazu in der Anwendung gängiger Testmethoden und Testwerkzeuge geschult ist. Da diese Mischung nur selten in einer Person anzutreffen ist, sieht sich der Testmanager gezwungen, ein gemischtes Testteam zu bilden, das aus Entwicklern, Analytikern, Anwendern und professionellen Testern besteht.

Oft umfassen Tests auch monotone Tätigkeiten mit Wiederholungscharakter, wie zum Beispiel das Erfassen Hunderter Testdatensätze oder die fünfte Wiederholung ein und desselben simplen Testfalles. Wo die Testautomatisierung noch nicht entsprechend fortgeschritten ist, benötigt man auch Personal, das von außen rekrutiert werden kann, zum Beispiel über eine Personal-Leasing-Agentur, oder in Form von Studenten der lokalen Universität. Man sollte dies aber nur als vorübergehende Lösung betrachten. Angestrebt ist die Automatisierung des Tests [FEWS99].

4.6 Testrisikoanalyse

Der Systemtest ist, wie andere IT-Unterfangen, nicht ohne Risiken. Es kann immer etwas vorkommen, was den Testtermin gefährdet. Die häufigsten Risiken:

- Die vorliegenden Anforderungsdokumente sind mangelhafter als erwartet.
- Die zu testende Software wird nicht rechtzeitig geliefert.
- Die Testwerkzeuge funktionieren nicht wie erwartet.
- Die geforderten Rechnerressourcen stehen nicht rechtzeitig bereit.
- Die Speicherkapazität reicht nicht aus.
- Performanzengpässe treten auf.
- Wichtiges Personal fällt aus.
- Die Software ist um einiges fehlerhafter als erwartet, so dass der Test mehrmals wiederholt werden muss [COHE04].

Jedes Risiko kann das Projekt verzögern, und mehrere Risiken zusammen können den Aufwand und die Dauer des Tests vervielfachen. Planen heißt auch vorbauen. Der Testmanager muss bei der Terminzusage einen Puffer vorsehen, um in der Lage zu sein, eventuelle Risiken zumindest teilweise aufzufangen. Für die Schätzung der Pufferzeit empfiehlt es sich, einen Risikozuschlag zu berechnen. Dafür muss jedes einzelne Risiko mit der Risikowahrscheinlichkeit RW und der Risikoaussetzung RA multipliziert werden. Die Risikowahrscheinlichkeit entspricht der Wahrscheinlichkeit in Prozent, dass dieses Risiko auftritt. Die Risikoaussetzung ist die prozentuale Zunahme der Projektdauer, zum Beispiel, dass die Zeitdauer um 25 % verlängert wird. Die Summe der Produkte dieser beiden Faktoren für alle Risiken ergibt den Gesamtrisikofaktor, zum Beispiel 0,64. Dieser wird mit eins addiert und mit der geplanten Testdauer in Wochen oder Monaten multipliziert, um die potenzielle Gesamttestdauer unter der Berücksichtigung der Risiken zu errechnen.

$$\text{Gesamttestdauer} = (1 + \Sigma\ \text{Risiko}\ (\text{RA} \times \text{RW})) \times \text{geplante Testdauer}$$

Bei einem Webprojekt hat einer der Autoren als Testplaner eine Zeitverschiebung von 3 Wochen vorgesehen (siehe Abbildung 4.8). Diese Verschiebung ist tatsächlich eingetreten und zwar genau um 3 Wochen. Es obliegt dem Testmanagement, risikomindernde Maßnahmen mit deren Hilfe die Zeitverschiebung reduziert werden könnte, vorzuschlagen. Beispiele dafür wären:

- Reservetester
- zusätzliche Rechenressourcen
- weitere Testwerkzeuge usw.

Der Testmanager hat jedoch auf viele Risiken wie die Fehlerhäufigkeit, die Lieferpünktlichkeit und die Performanzengpässe keinen Einfluss. Er kann sie allenfalls zu Protokoll geben, um sich später, wenn sie eintreten, zu rechtfertigen. Denn nach Murphys Law werden sie eintreten. Ein Softwaretester, und erst recht ein Softwaretestmanager, muss von Natur aus pessimistisch sein und trotz aller Widrigkeiten des Testgeschäfts die Fassung bewahren.

Die offensichtlichen Risiken des Tests sind:

- Verschiebungen der Lieferungen mit einer Wahrscheinlichkeit von 40%
- Hohe Fehlerhaftigkeit der gelieferten Software mit einer Wahrscheinlichkeit von 30%
- Technische Probleme mit der Testumgebung mit einer Wahrscheinlichkeit von 25%
- Probleme mit den Testwerkzeugen mit einer Wahrscheinlichkeit von 20%
- Ausfall wichtiger Projektmitarbeiter mit einer Wahrscheinlichkeit von 15%

- Die Auswirkung der Terminverschiebung auf die Testdauer könnte zwischen 2 bis 12 Wochen liegen.
- Die Fehlerhaftigkeit der Software könnte 1 bis 6 Wochen kosten.
- Technische Probleme mit der Testumgebung können 1 bis 4 Wochen kosten.
- Probleme mit den Testwerkzeugen können ebenfalls 1 bis 4 Wochen kosten.
- Der Ausfall von Personal kann das Projekt nicht mehr als 2 Wochen zurückwerfen, da genügend Reserven vorhanden sind und Tester sich mit geringem Aufwand ersetzen lassen.

Eine Analyse dieser Risiken führt zu folgendem Risikozuschlag:

Terminverschiebung	= [Max – Min Zeitverlust]/2 = 5 Wochen * 0,4 = 2 Wochen
Fehlerhaftigkeit	= [Max – Min Zeitverlust]/2 = 2,5 Wochen * 0,3 = 0,75 Wochen
Testumgebung	= [Max – Min Zeitverlust]/2 = 1,5 Wochen * 0,25 = 0,38 Wochen
Werkzeugprobleme	= [Max – Min Zeitverlust]2 = 1,5 Wochen * 0,20 = 0,30 Wochen
Mitarbeiterausfall	= [Max – Min Zeitverlust]/2 = 1 Woche * 0,15 = 0,15 Wochen

Der maximale Zeitverlust wäre die Summe aller Höchstgrenzen = 28 Wochen, bzw. 7 Monate.
Der wahrscheinlichste Zeitverlust wäre die Summe der einzelnen Risikofaktoren = 3,58 Wochen

ABBILDUNG 4.8 Beispiel einer Testrisikoanalyse

4.7 Festlegung der Testendekriterien

Eine unvermeidliche Aufgabe eines jeden Testprojektleiters ist es, die Testendekriterien festzulegen. Hier geht es um die schwierige Frage, wann der Test zu Ende ist. Es gibt Projekte, die ewig dahin laufen, weil niemand entscheiden kann, ob der Test ausreichend ist. Diesen Zustand muss man von vornherein vermeiden. Dazu braucht das Testprojekt messbare Endekriterien. Vorneweg ist festzustellen, dass sich kein komplexes Softwaresystem jemals vollständig testen lässt. Auch wenn es theoretisch möglich wäre, würde es zu lange dauern und zu viel Geld kosten. Darüber wurde schon genügend geschrieben [MUSA89]. Software ist wie das Universum, in dem wir leben, nach Einstein prinzipiell endlich, aber für alle praktischen Zwecke endlos. Ergo können wir einen vollständigen, 100%igen Test von Anfang an ausschließen. Umso mehr brauchen wir realistische Ziele für unseren Systemtest. Ein solches Ziel wäre die Fehlerüberdeckung. Aufgrund der Erfahrung mit ähnlichen Projekten aus der Vergangenheit wird die Anzahl der Fehler hochgerechnet. Man nimmt die alte Anzahl der Fehler und justiert sie durch die Größe und Komplexität des neuen Systems. Hatte das letzte Projekt dieser Art zum Beispiel 400 gemeldete Fehler bei einer Größe von 5000 Function-Points und einer Systemkomplexität von 0,54, wären beim neuen System, das auf 6000 Function-Points mit einer Komplexität von 0,6 geschätzt wird, 533 Fehler zu erwarten. Zunächst justieren wir die Größe des letzten Systems durch seine Komplexität.

$$\text{gewichtete Größe}_{\text{alt}} = 5000 \times \frac{0{,}54}{0{,}50} = 5400$$

Diese Größe dividieren wir durch die Anzahl der Fehler, um die Fehlerdichte zu ermitteln.

$$\text{Fehlerdichte} = \frac{400}{5400} = 0{,}074$$

Im nächsten Schritt justieren wir die geschätzte Größe des neuen Systems durch die geschätzte Komplexität.

$$\text{gewichtete Größe}_{\text{neu}} = 6000 \times \frac{0{,}6}{0{,}5} = 7200$$

Diese justierte Systemgröße wird jetzt mit der Fehlerdichte des letzten Systems multipliziert, um die erwartete Fehlerzahl zu errechnen.

$$\text{erwartete Fehleranzahl} = 7200 \times 0{,}074 = 533 \text{ Fehler}$$

Das eigentliche Testendekriterium wäre dann so lange zu testen, bis ein gewisser Prozentsatz der geschätzten Fehler aufgedeckt wird. Diese geschätzte Fehlerzahl muss auch nicht starr bleiben. Sie sollte dynamisch angepasst werden. Wenn der Testmanager beim Test der ersten Teilsysteme oder der ersten Version feststellt, dass die Fehlerdichte höher ist, zum Beispiel 0,09 statt 0,074, kann er die Anzahl der erwarteten Fehler hochschrauben, zum Beispiel von 533 auf 648.

Fehlerüberdeckung ist nicht das einzige Endekriterium. Es gibt auch die Abdeckung der Funktionalität, die Abdeckung der Daten oder die Abdeckung des Codes. Daraus folgen die drei Überdeckungskriterien:

- Funktionsüberdeckung
- Datenüberdeckung
- Codeüberdeckung

Funktionsüberdeckung ist der Prozentsatz, zu dem die spezifizierten Funktionen getestet werden. Aus der Anforderungsanalyse geht hervor, welche Anwendungsfälle mit welchen Aktionen und unter welchen Bedingungen es zu testen gilt. Ein messbares Testziel wäre es, einen vereinbarten Prozentsatz jener Funktionen auszuführen, zum Beispiel 90 %.

Datenüberdeckung ist der Prozentsatz, zu dem die spezifizierten Datenattribute in den Datenbanken, Systemschnittstellen und Benutzeroberflächen getestet werden. Für jedes zu testende Attribut sollte es eine Assertion geben, in der die Pre- und Postwerte des Attributs spezifiziert sind. Die Datenüberdeckung ist demnach der Prozentsatz der ausgeführten Assertions relativ zur Anzahl aller Attribute. Hier lässt sich auch ein gewisser Prozentsatz als Testendekriterium vereinbaren.

Schließlich gibt es die altbekannten Codeüberdeckungsmaße. Es sollte nicht das Ziel eines Systemtests sein, 90 % Zweigüberdeckung zu erreichen. Bei einem System mit einem hohen Anteil an wiederverwendetem Code, wird ein Großteil des Codes nie benutzt. Er ist nur deshalb da, weil es zu schwierig ist, den benutzen Code vom unbenutzten zu trennen. Die Codeüberdeckung ist eher ein Ziel des Unittests [BIND99]. Was jedoch gemessen werden kann, ist die Modul- bzw. die Methodenüberdeckung. Wie dies zu bewerkstelligen ist, wird erst später bei der Testauswertung behandelt. Hier genügt es, einen gewissen Modulüberdeckungsgrad als Testziel zu vereinbaren und dies auch konsequent anzustreben. Wenn er erreicht ist, ist das Testendekriterium erfüllt (siehe Abbildung 4.9).

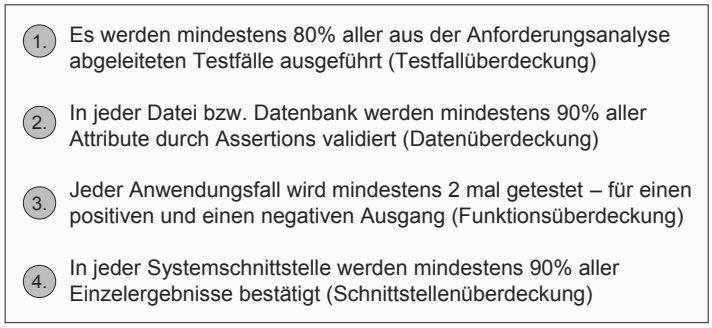

ABBILDUNG 4.9
Mögliche Testendekriterien

4.8 Gestaltung des Testplans nach ANSI/IEEE-829

Der letzte Akt der Testplanung ist die Verfassung des Testplandokuments. Bis dahin soll der Testprojektleiter, genauso wie jeder Autor einer wissenschaftlichen Arbeit, alle Fakten zusammengetragen haben.

- Er hat die Testziele gesetzt.
- Er hat die Testobjekte und Testfunktionen erkannt.
- Er hat die erforderlichen Soll-Testfälle gezählt.
- Er hat die Testressourcen identifiziert.
- Er hat den Testaufwand und die Testdauer geschätzt.
- Er hat die Testrisiken analysiert.
- Er hat einen Personenplan aufgestellt.
- Er hat die Testendekriterien aufgestellt.

Letzteres ist eine unbedingte Voraussetzung für die Testplanung. Der Testprojektleiter muss noch mit dem Gesamtprojektleiter und dem zuständigen Produktmanager darüber einig werden, wann der Test ausreichend ist bzw. wann man damit aufhören darf. Einige mögliche Endekriterien sind im letzten Abschnitt vorgestellt worden. Sie reichen von einer gewissen Code-, Daten- oder Funktionsüberdeckung bis hin zur Findung einer bestimmten Anzahl an Fehlern. Hinzu kommen die Einschätzungskriterien, wie die Errechnung der Budgetgrenze und die Überschreitung des Endtermins. Wichtig ist, dass die Verantwortlichen unter sich einig sind, welche Kriterien für diesen Test gelten sollen. Wenn dies feststeht, kann mit der Verfassung des Testplans begonnen werden. Als Muster für den endgültigen Testplan sollte das Gliederungsschema nach dem ANSI/IEEE-Standard 829 verwendet werden und zwar nicht nur, weil dieser die herrschende internationale Norm ist, sondern auch, weil er sich in der Praxis bewährt hat und vielfach zitiert wird [IEEE829]. Dieser Standard sieht ein Dokument mit 16 Abschnitten vor (siehe Abbildung 4.10).

ABBILDUNG 4.10 Aufbau des Testplans (nach ANSI/IEEE-Standard 829)

4.8.1 Testkonzept-ID

Hierbei handelt es sich um ein eindeutiges Kennzeichen des Dokuments, um es einerseits mit einer Suchanfrage wieder zu finden und andererseits von anderen Dokumenten aus zu referenzieren. Im Falle eines XML-Dokuments ist es das Attribut „ID“.

4.8.2 Einführung

Dieser Abschnitt ist eine kurze Zusammenfassung der zu testenden Hard- und Softwareeigenschaften und eine Zielerklärung, was mit dem Test erreicht werden soll, zum Beispiel, welche Risiken zu vermeiden sind. Falls es mehrere hierarchisch geschachtelte Testpläne gibt, muss jeder untergeordnete Testplan den übergeordneten referenzieren. In der Einleitung soll auch etwas über den Umfang des zu testenden Systems stehen. Hier soll es auch Hinweise auf für den Test relevante Dokumente geben. Diese sind:

- Spezifikationsdokumente bzw. das Fachkonzept
- Entwurfsdokumente bzw. das technische Konzept
- Benutzerhandbücher bzw. die Bedienungsanleitung
- Betriebshandbücher
- Installationsunterlagen

4.8.3 Zu testende Objekte

Hier sind sämtliche zu testende Objekte – Oberflächen, Schnittstellen, Datenbanken und Programme einschließlich der Versionsnummer bzw. Revisionsnummer – aufzuführen. Ausdrücklich nicht zu testende Objekte sollten ebenfalls aufgeführt werden.

4.8.4 Zu testende Funktionen

Hier wird die Benennung sämtlicher zu testender Funktionen und Funktionskombinationen verlangt. Falls das Anforderungsdokument nach Anwendungsfällen gegliedert ist, sind die Anwendungsfälle die zu testenden Funktionen. Für jede zu testende Funktion bzw. Funktionskombination sollte es einen Querverweis auf das dazugehörige Testskript oder die Testfallbeschreibung geben. In einem XML-Dokument wird dies das Attribut „href“ sein. In der Praxis kann diese Liste sehr lang werden, vor allem bei großen Systemen.

4.8.5 Nicht zu testende Funktionen

Besonders zu erwähnen sind jene Funktionen, die – aus welchen Gründen auch immer – nicht zu testen sind. Sie sind vom Systemtest ausgeklammert. Mit dieser Ausklammerung entzieht sich das Testteam der Verantwortung für die Funktionsfähigkeit dieser Funktionen bzw. erzwingt eine Festlegung an anderer Stelle, wer für die Qualitätssicherung der Funktionen verantwortlich sein sollte.

4.8.6 Testvorgehensweise

An dieser Stelle erfolgt eine Beschreibung des allgemeinen Testvorgehens bzw. der Teststrategie. Die Testphasen und Testaktivitäten sind hier anzugeben. Auch die einzusetzenden Techniken und Testwerkzeuge sind zu spezifizieren. Das Vorgehen sollte so weit beschrieben werden, dass es möglich ist, die einzelnen Testaufgaben zu identifizieren, zu schätzen und zuzuteilen. Außerdem sind alle bedeutenden und bekannten Einschränkungen des Testvorgangs aufzuführen, wie zum Beispiel Verfügbarkeit der Testobjekte, Ressourcen und Termine.

4.8.7 Testendekriterien

Hier geht es um Messkriterien für die Beendigung des Tests bzw. um die Frage, wann der Test fertig ist. Es müssen konkrete, messbare Ziele gesetzt werden, Ziele wie das Erreichen eines bestimmten Testüberdeckungsgrades, die Aufdeckung einer bestimmten Anzahl an Fehlern oder die Ausführung einer bestimmten Anzahl der Testfälle. Die Erfüllung dieser Endekriterien muss mit einem einfachen Ja oder Nein zu beantworten sein. Entweder ist das Kriterium erfüllt oder nicht. Nur wenn das Kriterium erfüllt ist, kann der Test als beendet betrachtet werden. Sonst gilt er als nicht abgeschlossen oder abgebrochen.

4.8.8 Testabbruchkriterien

An dieser Stelle sind jene Kriterien zu beschreiben, die eine Unterbrechung aller oder eines Teils der Testaktivitäten bedingen. Typische Abbruchkriterien:

- Schwerwiegende Fehler, die den Test behindern
- Zu viele Fehler
- Ungenügende Hardwarekapazitäten
- Die vorausgesetzten Softwareprodukte funktionieren nicht.
- Die Frist für den Test ist überschritten.

Falls ein oder mehrere dieser Kriterien auftreten, wird der Test zunächst unterbrochen. Wieder aufgenommen wird er erst, wenn die Probleme behoben sind, oder wenn eine Umgehungsmöglichkeit besteht. Hier kann auch festgelegt werden, unter welchen Bedingungen ein Test fortzusetzen ist.

4.8.9 Testergebnisse

Die Testergebnisse sind jene Produkte, die das Systemtestteam im Laufe des Tests zu liefern hat. Daran wird der Fortschritt des Tests gemessen. Die Liste beginnt mit dem Testplan und endet mit dem Testabschlussbericht. Die von der Norm vorgeschlagenen Ergebnisse sind:

- Testplan
- Testentwurf
- Testfallspezifikation
- Testskripte
- Testausführungsprotokolle
- Fehlerberichte
- Testabschlussbericht

Weitere Ergebnisse können hinzukommen, doch sind diese Produkte das Minimum dessen, was in einem Testprojekt erstellt werden sollte.

4.8.10 Testaufgaben

Neben den vom Test zu produzierenden Ergebnissen folgen hier die Aufgaben, die der Test zu erledigen hat. Es handelt sich um alle Tätigkeiten, die zur Vorbereitung, zur Ausführung und zur Auswertung des Tests erforderlich sind. Typische Testaufgaben:

- Erstellung des Testplans
- Entwurf eines Testkonzepts
- Spezifikation der Testfälle
- Generierung der Testskripte
- Bereitstellung der Testdaten

- Aufbau der Testumgebung
- Durchführung diverser Tests
- Protokollierung der Tests
- Fehlerberichterstattung
- Fehlerverfolgung
- Verfassung der Testberichte

Dies sind nur die wichtigsten Testaufgaben. In der Praxis wird es etliche Ausprägungen dieser Aufgabentypen geben. Man sollte Aufgaben definieren, die eine oder höchstens zwei Personen in wenigen Tagen erledigen können. Je feiner die Aufgaben definiert sind, umso leichter ist es, sie zu kontrollieren.

4.8.11 Testumgebung

Hier werden die erforderlichen Eigenschaften der Testumgebung festgelegt. Neben den gewünschten Hardwaregeräten wie Testarbeitsplätzen, Vermittlungsrechnern und Servern, Druckern und Speichergeräten werden die nötigen Softwareprodukte angegeben: Betriebssysteme, Datenbanksysteme und Middleware sowie Office-Produkte und spezielle Frameworks. Sollten Testwerkzeuge benötigt werden, sind diese auch hier zu beschreiben. Ebenso werden hier räumliche Anforderungen gestellt, so zum Beispiel zusätzliche Testräume mit Netzanschlüssen oder Funkverbindungen. Für alle Materialien, die im Moment noch nicht zur Verfügung stehen, ist eine Quelle zu nennen.

4.8.12 Testverantwortlichkeiten

Hier sind die verantwortlichen Gruppen bzw. Dienststellen für Management, Design, Vorbereitung, Ausführung, Beglaubigung, Überprüfung und Support zu definieren. Soweit bekannt, werden die einzelnen Testaufgaben verantwortlichen Personen oder Stellen zugewiesen. Es geht also auch um die Organisation des Testprojekts und die Zuteilung der Rollen.

4.8.13 Testpersonalbedarf

Ausgehend von den Rollen im Testverfahren wird hier ein Personalbedarfsplan aufgestellt. Die verschiedenen Rollen setzen gewisse Fähigkeiten voraus, zum Beispiel die des Datenbankspezialisten, des Netzspezialisten, des Sicherheitsexperten oder des XML-Kenners. Zusätzlich zu diesen Spezialkenntnissen wird ein Ausbildungsplan für das Testpersonal vorgelegt, damit die Tester besser in der Lage sind, ihre Rollen zu erfüllen. Dazu gehören die Schulung der Testmethodik und der Bedienung der Testwerkzeuge sowie Schulungen bezüglich der Testobjekte, der zu testenden Applikationen selbst.

4.8.14 Testzeitplan

Im Zeitplan werden die Meilensteine im Testprojekt sowie die Liefertermine der Testobjekte festgehalten. Auch zusätzliche Meilensteine können aufgeführt werden. Für jede Testaufgabe ist der Zeitbedarf zu schätzen, damit sich ein Netzplan oder PERT-Diagramm erstellen lässt, aus dem der kritische Pfad hervorgeht. Zur Darstellung des Zeitplans können konventionelle Projektmanagementmittel – wie GANTT-Diagramme – herangezogen werden. Aufgrund des Testplans ist schließlich für jede Testressource – Hardware, Software oder Personal – festzuhalten, wann sie zu welchem Zweck gebraucht wird.

4.8.15 Testrisiken und Risikomanagement

Jeder Systemtest ist mit Risiken behaftet. Es gibt vertragliche, technische, betriebliche und menschliche Risiken. Hier sind die einzelnen Risiken zu identifizieren und deren Auswirkung und Wahrscheinlichkeit abzuschätzen. Typische Testrisiken:

- Mängel in den Anforderungsdokumenten (Verfügbarkeit, Aktualität, Qualität)
- Fehlender Zugriff auf die Verfasser der Anforderungsdokumente
- Verzögerungen bei der Softwarelieferung
- Schwerwiegende Konstruktionsfehler in der Software
- Hardwareausfälle
- Performanzengpässe
- Instabile Grundsoftware
- Nichtfunktionierende Testwerkzeuge
- Personalprobleme

Für jedes mögliche Risiko ist eine Gegenmaßnahme vorzusehen, zum Beispiel: zusätzliche Hardwarekapazität beschaffen, den Softwarelieferanten Konventionalstrafen androhen, Personal austauschen, Überstunden anordnen und auf andere Werkzeuge ausweichen. Die Maßnahmen sind zu bewerten, inwieweit sie geeignet sind, Risiken zu mindern.

4.8.16 Genehmigungen

Am Ende des Testplans kommen die Namen und Titel aller Verantwortlichen, die diesen Testplan genehmigen müssen. Ihre Unterschrift bezeugt, dass sie mit dem Inhalt des Testplanes einverstanden sind. Zu Dokumentationszwecken sollte eine gedruckte Kopie des Testplans unterzeichnet und archiviert werden. Es empfiehlt sich, die elektronische Version des Testplans zu speichern und bereitzustellen. So kann er jederzeit wieder abgerufen und mit den anderen Dokumenten des Projekts leichter verbunden werden.

Im Gegensatz zu vielen anderen Normen, die eher akademischer Natur sind, ist der IEEE Standard 829 sehr praxisbezogen. Er wird nicht nur von der Firma ANECON, sondern auch von etlichen anderen Testfirmen, wie z. B. der SQS in Köln und der IFS Consulting in Eschborn, praktiziert. Die IFS hat einen interessanten Erfahrungsbericht über die Umsetzung des

Standards im GI Software Management-Rundbrief veröffentlicht [SLAD05]. Anhang A dieses Buches enthält einen Testplan aus der ANECON-Testpraxis.

4.9 Die Prüfspezifikation nach V-Modell-XT

In Projekten für den öffentlichen Dienst wird kein Testplan, sondern eine Prüfspezifikation verlangt. Eine Prüfspezifikation unterscheidet sich von einem Testplan hauptsächlich in der Terminologie. Statt vom Testen ist von Prüfen die Rede weil Prüfen ein umfassenderer Begriff ist. Er umfasst auch die verschiedenen Formen des statischen Testens wie Reviews, Inspektionen und Analysen. Die Analyse der Anforderungsdokumente durch den Textanalysator sowie die Inspektion des Codes durch einen CodeAuditor sind Prüfungen, aber kein Test. Test impliziert die dynamische Ausführung einer Software. Insofern ist der deutsche Begriff „Prüfen“ weiter gefasst. Die Prüfspezifikation aus dem V-Modell-XT schließt sämtliche Prüfaktivitäten ein, sowohl die Prüfung der Dokumente als auch den Test der Programme [UFAB05]. Der Testplan aus dem amerikanischen IEEE-Standard bezieht sich ausschließlich auf den Systemtest. Das Problem besteht darin, dass in Projekten der öffentlichen Hand Prüfspezifikationen abgegeben werden müssen. Ergo muss der Inhalt des IEEE-Testplanes in das Format einer V-Modell-XT-Prüfspezifikation gegossen werden. Dies ist, wie die folgende Inhaltsbeschreibung zeigt, einigermaßen gelungen. Es ist sogar möglich, aus der Anforderungsanalyse eine Prüfspezifikationsvorlage mit einigen vorgefertigten Inhalten wie den zu testenden Objekten und Funktionen sowie die Aufwandsschätzung automatisch zu generieren.

ABBILDUNG 4.11 Aufbau der Prüfspezifikation (nach V-Modell-XT)

4.9.1 Einleitung

Diese Prüfspezifikation beschreibt alle vorgesehenen Prüfaktivitäten im Rahmen des jeweiligen Projekts. Die hier beschriebenen Prüfaktivitäten beziehen sich sowohl auf den Abnahmetest des fertigen Systems als auch auf die Prüfung der Zwischenergebnisse, die laut Vertrag abzuliefern sind. Zur Abnahme des fertigen Systems gehört die Planung, Vorbereitung, Organisation, Durchführung und Auswertung des Gesamtsystems. Getestet wird das System gegen die funktionalen und nicht-funktionalen Anforderungen des Lastenheftes. Laut V-Modell-Vorschrift darf nur das geprüft werden, was das Lastenheft explizit vorschreibt. Deshalb handelt es sich hier um einen anforderungsbasierten Test. Es gilt, jede einzelne Anforderung auf seine Erfüllung zu prüfen, ehe das Produkt abgenommen wird. Dazu gehört eine ganze Reihe von Prüffällen. Neben den funktionalen und nicht-funktionalen Anforderungen sind die in diesem Dokument identifizierten Risiken zu prüfen.

Für die Prüfung der abgelieferten Dokumente sind Reviews und Inspektionen vorzusehen, die von der betrieblichen Qualitätssicherung zu überwachen sind. Für die Prüfung des abgegebenen Source Codes ist eine toolgestützte statische Analyse durchzuführen, bei der die Sourcen gegen die vereinbarten Codierregeln geprüft und anhand der vereinbarten Quantitäts-, Komplexitäts- und Qualitätsmetriken gemessen werden. Es handelt sich hier um projektbegleitende Qualitätssicherung und Abnahmetests in Anlehnung an die Vorgaben des QS-Handbuches.

4.9.2 Prüfziele

Das Prüfziel besteht darin, die Ist-Eigenschaften des gelieferten Systems mit den Soll-Eigenschaften laut Lastenheft abzugleichen und die Differenz zu dokumentieren. Dazu gehören folgende Soll/Ist-Abgleiche:

- Abgleich der Soll/Ist-Funktionalität
- Abgleich der Soll/Ist-Performanz
- Abgleich der Soll/Ist-Benutzbarkeit
- Abgleich der Soll/Ist-Zuverlässigkeit
- usw.

In diesem Abschnitt werden die Ziele der Prüfungen eindeutig festgelegt, so dass sie im gleichen Maße für Auftraggeber und Auftragnehmer verständlich sind.

4.9.3 Prüfobjekte

Die Objekte der Prüfung:

- Pflichtenheft
- Implementierungsplan
- Entwurfsdokumente
- Source Programme

- Benutzeroberflächen
- Berichte
- Datenbanken
- Schnittstellen
- Subsysteme

In diesem Abschnitt werden die einzelnen Prüfobjekte aufgezählt und kurz beschrieben.

4.9.4 Prüffälle

Die Prüffälle sind nach ihrer Herkunft zu klassifizieren. Sie ergeben sich zum Teil aus dem Lastenheft, zum Teil aus der Risikoanalyse und zum Teil aus der Erfahrung mit früheren Projekten dieser Art. Daraus folgen drei Arten von Prüffällen:

- Anforderungsbasierte Prüffälle (Funktional und Nicht-Funktional)
- Risikobasierte Prüffälle
- Erfahrungsbasierte Prüffälle

An dieser Stelle sind sämtliche aus der Anforderungsanalyse gewonnenen Prüffälle aufzulisten. Hinzu kommen die Prüffälle aus der Risikoanalyse sowie jene aus der Erfahrung des Testers. Da es in der Regel mehrere Hundert Prüffälle gibt, empfiehlt es sich, sie in eine Tabelle im Anhang zur Prüfspezifikation auszulagern. Hier werden die Prüffallarten identifiziert, und es wird auf sie hingewiesen.

4.9.5 Prüfstrategie

Hier geht es darum, die Prüf- bzw. Teststrategie zu erläutern. Die Teststrategie ist mit der Testvorgehensweise im IEEE-Testplan zu vergleichen. Zusätzlich folgt eine Vorgehensweise für die Prüfung und Abnahme der Dokumente, die der Auftraggeber liefert. Dazu gehört das Pflichtenheft, der Implementierungsplan, die Entwicklertestdokumentation usw. Der Auftraggeber benötigt eine Strategie, wie er sie zu prüfen gedenkt.

4.9.6 Prüfkriterien

In diesem Abschnitt sind die Kriterien für die Abnahme der Zwischen- und Endergebnisse aufzuzählen. Falls man die Dokumente und die Programm-Sourcen gegen Richtlinien prüft, muss man hier auf die Richtlinien verweisen. Für den Abnahmetest werden die Testendekriterien spezifiziert – Kriterien wie Funktionsüberdeckung, Erfüllung der Anforderungen und Restfehlerwahrscheinlichkeit. Diese Kriterien sind vertraglich verbindlich und sollten deshalb juristisch unanfechtbar formuliert werden. Zu den Prüfkriterien gehören auch die Abbruchkriterien aus dem IEEE-Testplan, d. h. die Bedingungen, unter denen der Abnahmetest abgebrochen wird, z. B. wenn die Fehlerrate eine bestimmte Grenze überschreitet.

4.9.7 Prüfergebnisse

Bei den Prüfergebnissen kommen zu den Testberichten auch die Protokolle der Dokumentenprüfungen. Der Test als solcher liefert die Prüffälle, den Testentwurf, die Testprotokollierung, den Abnahmetestbericht und die Fehlerberichte. Die Prüfungen liefern Mängelberichte und Abnahmeprotokolle sowie Verbesserungsvorschläge, also die üblichen Ergebnisse.

4.9.8 Prüfaufgaben

Die Prüfaufgaben folgen aus den zu liefernden Prüfergebnissen. Für jedes Prüfergebnis gibt es eine Prüfaufgabe, die das Ergebnis erzeugt. Typische Prüfaufgaben in einem V-Modell-XT-Projekt sind:

- Prüfung des Pflichtenheftes
- Prüfung des Implementierungsplanes
- Prüfung der Entwurfsdokumente
- statische Analyse des Source-Codes
- Prüfung der Entwicklertestdokumentation
- Entwurf des Abnahmetests
- Spezifikation der Testfälle
- Erstellung der Testdaten
- Aufbau der Testumgebung
- Durchführung des Abnahmetests
- Auswertung des Abnahmetests

4.9.9 Prüfumgebung

Die allgemeine Prüfumgebung wird bereits in den zugehörigen Implementierungs-, Integrations- und Prüfkonzepten beschrieben. In diesem Abschnitt beschreiben wir notwendige Ausgestaltungen und Erweiterungen der allgemeinen Prüfumgebung oder speziell notwendige Prüfumgebungen für das konkrete Prüfobjekt. Es werden Anforderungen sowohl an die Hardware als auch an die Software gestellt. Hier werden die erforderlichen Kapazitäten genau vorgegeben.

4.9.10 Prüffallzuordnung

Im originalen V-Modell-XT-Ton heißt es hier: „Die aus den Anforderungen abgeleiteten Prüffälle werden in einer Abdeckungsmatrix den Anforderungen zugeordnet. Hier soll sichtbar werden, ob der gewünschte Abdeckungsgrad und die Prüfqualität gegeben sind, besonders in Bezug auf die vorher festgelegte Prüfstrategie.“ Im Klartext bedeutet dies, dass die Prüffälle mit den Anforderungen in einer Tabelle vereint werden. Für die automatisch aus dem Lastenheft abgeleiteten Prüffälle erledigt dies bereits das Textanalysator-Tool, wie das folgende Beispiel verdeutlicht. Für die hinzugekommenen Prüffälle muss dies noch erfolgen. Es geht hier darum festzustellen, dass jede Anforderung, ob funktional oder nicht-funktional, durch mindestens einen Prüffall abgedeckt ist.

Zur Prüffallzuordnung gehört auch die Zuordnung der Prüffälle zu den verantwortlichen Prüfern bzw. Testern. Jeder Prüffall sollte einen Zuständigen haben, der darauf achtet, dass dieser Fall tatsächlich geprüft und abgenommen wird. Dies wird ebenfalls von dem Textanalysator-Werkzeug unterstützt. Sofern der Verfasser des Lastenheftes die Anforderungen jemandem zuordnet, wird dem Verantwortlichen für die Anforderung auch die Verantwortung für die Prüffälle dieser Anforderung übertragen.

Anforderung	Testfall
FUNC-REQ 1_Artikelanzeige	AUFTRAG0001
FUNC-REQ 1_Artikelanzeige	AUFTRAG0002
FUNC-REQ 2_Artikelbestellung	AUFTRAG0003
FUNC-REQ 2_Artikelbestellung	AUFTRAG0004
FUNC-REQ 2_Artikelbestellung	AUFTRAG0005
FUNC-REQ 2_Artikelbestellung	AUFTRAG0006
FUNC-REQ 3_Versandauftrag	AUFTRAG0007

ABBILDUNG 4.12 Zuordnung von Anforderungen zu Testfällen

4.9.11 Prüfaufwand

An dieser Stelle wird der geschätzte Aufwand für die Prüfungsaktivitäten vorgegeben. Diese Aufwandsschätzung ist besonders wichtig, wenn der Auftrag zur Qualitätssicherung des Projekts ausgeschrieben wird, aber auch wenn der Auftraggeber die Prüfaufgaben selber übernimmt, sollte er wissen, wie viel Kapazität er dafür vorsehen muss. Der Textanalysator erzeugt diese Schätzung automatisch aufgrund der rohen Anzahl an Test-Points, wenn er das Prüfspezifikationsmuster generiert. Das folgende Beispiel wurde automatisch generiert.

```
Der Aufwand für die Vorbereitung, Ausführung und Auswertung des Abnahmetests
setzt sich aus folgenden Posten zusammen:
     Personalkosten = Anzahl Testpunkte / Testproduktivität
     Anzahl Testfälle = 409
     Anzahl Test-Points = 685
     Testproduktivität = 8 Test-Points pro Testtag
     685 / 8 = 86 Testertage
```

4.9.12 Risikovorkehrungen

Für jedes Prüfobjekt mit einem Gefährdungspotenzial bei der Prüfung, das nicht normal getestet werden kann, wird beschrieben, welche Vorkehrungen und Maßnahmen durchzuführen sind, damit bei seiner Prüfung keine Gefährdungen auftreten können. So heißt es im originalen V-Modell-XT-Text. Diese Vorsichtsmaßnahmen gelten für die Prüfung von Geräten. Für die Prüfung von IT-Systemen gilt es hier die Gegenmaßnahmen für eventuelle Risiken beim Test festzuschreiben. Solche Risiken können zur Verschiebung bzw. zum Abbruch des Tests führen. Somit haben sie eine Auswirkung auf das Gesamtprojekt. Zum Abfangen jener Ereignisse sind Vorkehrungen zu treffen.

Die Prüfspezifikation endet mit den üblichen Abkürzungs- und Literaturverzeichnissen. Wie aus dieser kurzen Inhaltsschilderung hervorgeht, lässt sich die Prüfspezifikation aus dem V-Modell-XT durchaus mit dem Testplan aus dem ANSI/IEEE-Standard kombinieren. Die wesentlichen Inhalte des Testplanes passen dort gut hinein. Von Fall zu Fall sollte man den Unterschied zwischen Prüfen und Testen im Auge behalten.