Kapitel 1
Überblick

Der Beruf der Softwareentwicklerin bzw. des Softwareentwicklers ist sehr vielfältig. Neben der Bewältigung technischer Herausforderungen sind Teamarbeit, Kommunikation und eine strikte Lösungsorientierung wichtig. Unser Buch wird Sie auf dem Weg zum Profi begleiten. Kommen Sie mit auf eine spannende Reise.

Dieses Fachbuch setzt sich umfassend mit den vielfältigen Themen der Softwareentwicklung auseinander. Sie erfahren alles Wichtige, um für Ihre Kundinnen und Kunden bestmögliche Programme erstellen zu können. Dabei werden Sie schnell feststellen, dass Softwareentwicklung viel mehr ist als nur die Beherrschung einer Programmiersprache. Heutige Software ist komplex und kommt in all unseren Lebensbereichen vor. Um komplexe Produkte zu entwickeln, braucht es Professionalität. Alle diese Fähigkeiten, wie technische Kompetenz, Teamfähigkeit und Lösungsorientierung, kommen in diesem hochinteressanten Beruf zusammen, den wir Ihnen zu Beginn dieses Kapitels vorstellen. Danach erfahren Sie, wie wir dieses Buch für Sie strukturiert haben.

1.1 Berufswunsch Softwareentwickler

Wann haben Sie sich die ersten Gedanken zu Ihrer Berufswahl gemacht? Im Kindergartenalter? Das trifft auf die meisten von uns zu. Viele Jungs wollen zunächst Feuerwehrmann, Polizist oder Lokführer werden, Berufe mit vermeintlicher Action, mit Spannung und mit einer großen Außenwirkung. Und die Mädchen? Oft werden hier in jungen Jahren Balletttänzerin oder Tierärztin genannt. Die Zeit vergeht, und die Wunschberufe ändern sich! Dieses Buch ist ein Lehrbuch für angehende professionelle Softwareentwicklerinnen und -entwickler und zugleich hoffentlich ein gutes Nachschlagewerk für alle diejenigen, die sich diesen Berufswunsch schon erfüllt haben.

Für alle, die sich schon sicher sind, dass sie keine Tierärztin werden und dass auch der Beruf des Lokführers eher nicht der richtige zu sein scheint, ist dieses erste Kapitel gedacht. Vielleicht tragen Sie sich ja ernsthaft mit dem Gedanken, die Softwareentwicklung zu Ihrer Profession zu machen und damit künftig Ihre Brötchen zu verdienen. Die Berufsbezeichnung »Softwareentwickler« bzw. »Softwareentwicklerin« klingt

zunächst einmal spannend, und Sie werden sich vielleicht die folgenden Fragen stellen: Welche Aufgaben umfasst dieser Beruf? Stimmt das Klischee vom Nerd, der den ganzen Tag zwischen Unmengen von Pizzapackungen und schmutzigen Kaffeetassen einsam im dunklen Kämmerlein sitzt und dabei kryptische Zeichen in den Rechner hackt? Welche Ausbildung braucht man, um Softwareentwickler oder -entwicklerin zu werden? Welche persönlichen Eigenschaften sollte man idealerweise mitbringen? Wie sind die Chancen auf dem Arbeitsmarkt? Und natürlich: Wie sieht es mit dem Gehalt aus? Wir möchten Ihnen einen Eindruck davon vermitteln, welche Erwartungen Sie in Bezug auf den Beruf hegen können und welche Anforderungen an Sie gestellt werden.

Wie die Berufsbezeichnung schon sagt: Die Hauptaufgabe dabei ist das Entwickeln und Erstellen von Software. Ist damit die Frage nach dem Berufsinhalt vollständig beantwortet? Nein! Der Beruf ist sehr vielfältig. Das Aufgabenspektrum hat sich in den letzten Jahren erheblich erweitert. Die Kernkompetenz kann man wie folgt umschreiben:

> *Softwareentwicklerinnen und Softwareentwickler analysieren, planen und implementieren auf professionelle Art und Weise informationstechnische Anwendungen und Softwarebausteine. Ebenso sind sie für umfassende Tests und für eine nahtlose Systemintegration zuständig.*

In diesen beiden Sätzen steckt die gesamte Vielfalt des Berufsbilds. Im Einzelnen: Eine wesentliche Aufgabe des Entwicklungsprozesses ist die umfassende Analyse der Aufgabe, d. h. die Überlegung, welche Art von Software erstellt werden soll. Welchen Anforderungen soll diese genügen? Was erwarten unsere Kundinnen und Kunden von der Anwendung? Welche Funktionen und welche Bedienoberfläche wünschen sich diejenigen, die die Anwendung künftig nutzen? Diese Fragen müssen wir uns, die wir entwickeln, stellen und sorgfältig beantworten. Dazu müssen wir mit unserer Kundschaft reden, ihr zuhören, weitere Fragen stellen und beobachten. Unser Beruf ist so gesehen mit einem Beruf in der Forschung zu vergleichen. Zufrieden sind Kundinnen und Kunden nur dann, wenn wir ihre Wünsche erfüllen. Um das zu tun, müssen wir sie erst einmal kennen.

Bevor wir ein Programm erstellen können, müssen wir diesen Prozess umfangreich planen. Vorbei sind die Zeiten, als der erste Schritt im Schreiben des *Quellcodes* bestand. Professionelle Entwicklung startet heutzutage viel früher. Nach der Analyse folgt die Planung. Diese wird man zwar später wieder überarbeiten müssen, aber wenn man am Schluss ein gutes Ergebnis erzielen will, gibt es am Anfang nicht viele Alternativen.

Implementierung meint dann den eigentlichen Vorgang der Programmerstellung, d. h. das Codieren in der jeweiligen Programmiersprache. Viel zu oft wird auch heute noch der Vorgang der Softwareentwicklung mit dem Teilschritt der Implementierung bzw. mit der Programmierung gleichgesetzt. Richtig ist, dass es ein wichtiger Teilschritt ist. Richtig ist aber auch: Es ist eben nur ein einzelner Teilschritt auf einer ganzen Liste von Aufgaben! Deshalb finden wir, dass die umfassende Aufgabe der Softwareentwicklung und die dazu notwendige hochprofessionelle Vorgehensweise besser durch den Terminus *Software Engineering* ausgedrückt wird. Software Engineering klingt nach einem geplanten und ingenieurgemäßen Vorgehen. Diesen Anspruch hat die heutige Entwicklung von Software auch, und damit grenzt sie sich eindeutig vom Programmieren als Hobby oder Zeitvertreib ab. Sie verwendet erprobte Methoden, setzt auf eine umfassende Werkzeugunterstützung, hat stets den Charakter eines Projekts und verfolgt ein klares Ziel. Dieses Ziel ist in vielen Fällen auch wirtschaftlicher Natur. Anwendungen werden in Auftrag gegeben und erstellt, um damit Probleme leichter lösen zu können und um letztendlich damit Geld zu verdienen.

Kehren wir noch ein wenig zum Prozess der Erstellung der Software zurück. Auch nach der vermeintlichen Fertigstellung der Software können wir uns noch nicht ausruhen, denn das Projekt ist noch nicht zu Ende gebracht. Funktioniert alles so, wie es soll? Haben wir die Wünsche unserer Kunden verstanden und auch komplett umgesetzt? Umfassende Tests gehören genauso zu unseren Aufgaben wie eine vollständige *Systemintegration*. Heutzutage kommt ein Programm selten allein daher. Es trifft so gut wie immer auf eine gewachsene IT-Infrastruktur. Dies bedeutet, dass das Programm in die jeweiligen IT-Systeme integriert werden muss. Um das zu tun, müssen wir umfassend mit unserer Kundschaft zusammenarbeiten.

Unser Aufgabenspektrum ist also ziemlich breit! Natürlich kann man am Ende nicht in allen Bereichen Expertin oder Experte sein, man muss sich spezialisieren. Trotz einer späteren Spezialisierung ist es wichtig, dass Sie gleich zu Beginn einen guten Überblick über alle Facetten des Berufs bekommen.

Nun aber sehen wir uns das Berufsbild noch etwas genauer an und grenzen es auch in der einen oder anderen Richtung von verwandten Jobs in der IT ab.

1.1.1 Sind Softwareentwickler besondere Informatiker?

Fragen Sie Ihre Oma, was man in der Softwareentwicklung so tut, dann lautet die Antwort vermutlich in etwa: »Irgendwas mit Computern.« Das stimmt, ist aber leider sehr ungenau, denn es trifft auch auf eine ganze Reihe weiterer Berufe aus dem Bereich der Informatik zu. Wenn Sie sich mit dem Berufsbild beschäftigen möchten, gehört es auch dazu, sich die verwandten Berufe und Tätigkeiten anzusehen. Sie werden feststellen, dass es Überschneidungen gibt und die Abgrenzungen nicht immer eindeutig sind.

Um auf Oma zurückzukommen: Mit Computern arbeiten heute fast alle, auch der Sachbearbeiter bei der örtlichen Finanzverwaltung oder die Automechanikerin um

die Ecke. In diesem Buch geht es jedoch darum, IT nicht nur als Hilfsmittel zu verwenden, sondern sie auch Anwendern bereitzustellen. In welchen Bereichen können Sie als IT-Spezialist heute arbeiten? Neben der Softwareentwicklung gibt es in verwandten Berufen etwas zu tun, wie etwa im Webdesign oder der Systemadministration. Klangvolle Berufsbezeichnungen in Stellenanzeigen zeugen vom Versuch, möglichst viele Interessierte anzusprechen. Die Kreativität der Branche kennt hier keine Grenzen. Nicht immer ist intuitiv klar, was genau inhaltlich dahintersteckt. Nachfolgend versuchen wir, diese unvollständige und persönliche Auswahl an IT-Berufen etwas näher zu beschreiben.

Beginnen wir mit dem Arbeitsfeld der IT-Spezialisten. – gewissermaßen der Überbegriff für Personen, die sich professionell mit der Informationsverarbeitung in Unternehmen beschäftigen. Diese Menschen können beispielsweise als Leiterin einer IT-Abteilung, als Entwickler oder als Systemadministratorin in einem Unternehmen arbeiten. Informatik kann man im Rahmen der dualen Ausbildung zum sogenannten Fachinformatiker erlernen oder in verschiedenen Formen an Berufsakademien, Fachhochschulen und Universitäten studieren. Das Aufgabengebiet ist später vom Unternehmen und der sich daraus ergebenden Spezialisierung abhängig. Die Anforderungen an diesen Beruf sind vielschichtig. Technisches Verständnis gehört genauso dazu wie die Fähigkeit, im Team zu arbeiten. Fazit: *Den* einen Informatiker gibt es nicht, lediglich die Ausbildung zu diesem Beruf. Eine Spezialisierung und Konkretisierung der Aufgaben ergibt dann der Berufsalltag. Wenn man kann, sollte man hier am besten seinen Interessen folgen.

Kommen wir zum Beruf des Webdesigners. *Webdesigner* und *Webdesignerinnen* sind Personen, die für die Gestaltung und Umsetzung von Webseiten aller Art verantwortlich sind. Auch diese Menschen müssen sich mit vielfältigen technischen Aspekten auseinandersetzen. Zu nennen sind beispielsweise *HTML*, *CSS*, *JavaScript*, *Webframeworks* und *PHP*. Die Liste könnten wir unendlich fortsetzen. Die Technik gilt es also zu beherrschen, und dazu ist auch wieder ein grundlegendes Verständnis von Informatik notwendig. Hinzu kommt aber noch eine weitere Anforderung, nämlich der Designaspekt, der bei diesem Beruf eine besonders große Rolle spielt. Webseiten sollen nicht nur funktionieren, sondern auch ansprechend gestaltet sein und die Arbeitsweise derjenigen, die sie verwenden, bestmöglich unterstützen. Webdesign-Profis müssen sich also in diesen nicht technischen Bereichen ebenfalls eine Menge Fähigkeiten erarbeiten. Das kann durch eine Ausbildung und auch durch *Learning by Doing* geschehen. Die Grenzen zwischen diesem Berufsbild und den Berufsbildern von Grafik- und Mediendesignern sind fließend. Mit zunehmender Nutzerorientierung sind Fragen des Designs auch im Bereich der Softwareentwicklung präsent. Eine Folge davon ist, dass immer mehr Softwareentwicklungsunternehmen neben Profis für die Prpogrammierung auch solche für das Design beschäftigen.

Und für die *Systemadministration*? Es gibt keinen einheitlichen Ausbildungsweg, der zu einem anerkannten Abschluss »Systemadministrator« führen würde. Dennoch gibt es in den IT-Abteilungen vieler Unternehmen Personen, deren hauptsächliche – oder sogar ausschließliche – Aufgabe es ist, dafür zu sorgen, dass die IT ständig verfügbar ist. Systemadministratorinnen und Systemadministratoren haben die Aufgabe, Software zu installieren, sie zu konfigurieren und die *Anwendungssysteme* untereinander kompatibel zu halten. In Unternehmen hat man es nicht mit einem einzelnen Programm zu tun, zur Lösung der anstehenden Aufgaben müssen viele Programme zusammenarbeiten. Systemadministratoren und -administratorinnen pflegen die Datenbanken und sorgen für Integration und Zusammenarbeit verschiedener IT-Systeme. Um diesen Beruf zu ergreifen, muss man sich allgemein sehr gut in der Informatik auskennen. Der Weg dahin kann beispielsweise über Ausbildungen zum Betriebsinformatiker oder zum Fachinformatiker führen. Alternativ ist natürlich auch ein Studium der Informatik möglich.

Neuere Entwicklungen aus den Bereichen zur Anwendung der Methoden der künstlichen Intelligenz und der Datenanalyse führen auch zu neuen Berufsbezeichnungen. Eine solche ist der Beruf des *Data Scientist* bzw. *Datenwissenschaftlers*. Die fortschreitende Digitalisierung in allen Bereichen unseres Lebens führt zur Produktion riesiger – fast unvorstellbarer großer – Datenmengen. Aus Daten die entsprechenden Informationen zu gewinnen und letztendlich daraus möglichst neues Wissen abzuleiten, ist das primäre Aufgabenfeld dieses Berufs. Kenntnisse der IT, der Statistik und der Mathematik machen diesen Beruf anspruchsvoll, aber auch gleichzeitig hoch spannend.

Kommen wir auf den in Hinblick auf dieses Buch wichtigsten Beruf unter allen, die auf IT spezialisiert sind – auf den Softwareentwickler bzw. die Softwareentwicklerin. Umgangssprachlich werden sie auch als Programmierer oder als *Developer* bezeichnet. In diesem Beruf spielen praktische Erfahrungen mindestens eine genauso wichtige Rolle wie ein formaler Abschluss einer Ausbildung oder eines Studiums! Ein netter Titel auf einem Blatt Papier befähigt die betreffende Person noch nicht dazu, in der Softwareentwicklung zu arbeiten. Neben umfassenden theoretischen Kenntnissen sind Praxiserfahrungen unerlässlich. Wenn Sie diesen Beruf ergreifen wollen, müssen Sie an zwei Dingen ein stetiges und sehr hohes Interesse haben:

1. **Interesse an Neuem**: In kaum einer Branche sind die Halbwertzeit von Wissen und die Aktualität von Technologien so kurz wie im Bereich der Softwareentwicklung. Um auf lange Zeit erfolgreich Software entwickeln zu können, muss man neugierig und neuen Technologien gegenüber aufgeschlossen sein und darf keine Angst vor Veränderungen haben. Die meisten Profis sind Feuer und Flamme, wenn es darum geht, eine neue Programmiersprache oder ein neues Tool auszuprobieren.

2. **Bereitschaft zum Lernen**: Genauso wichtig wie das Interesse an neuen Technologien ist die Motivation, sich kontinuierlich weiterzubilden. Technologien kommen und gehen, Arbeitsweisen verändern sich ständig, und neue Herangehensweisen wollen erlernt werden. Das viel zitierte lebenslange Lernen wird mittlerweile in etlichen Berufen zwar als Notwendigkeit benannt, in der Softwareentwicklung ist es gewissermaßen überlebensnotwendig. Interessant: Ein Großteil der Entwicklerinnen und Entwickler bezeichnet sich selbst als Autodidakten.

Wer professionell entwickelt, möchten mit seinem Beruf natürlich auch seinen Lebensunterhalt verdienen. Im nächsten Abschnitt gehen wir daher etwas detaillierter auf die Arbeitsmarktsituation ein.

1.1.2 Arbeitsmarktsituation und Verdienstmöglichkeiten

Wie sieht es auf dem Arbeitsmarkt aus? Welche Chancen gibt es und mit welchem Gehalt darf man rechnen?

Eines steht fest: Der Mangel an Fachkräften bestimmt die aktuelle Arbeitsmarktsituation für diesen Beruf. Was heißt das? Wer profitiert von diesem Mangel? Die Nachfrage ist derzeit hoch, und es sieht nicht so aus, als ob sich daran mittelfristig etwas ändern würde. Der Bedarf an ausgebildeten Profis für die Softwareentwicklung steigt aufgrund zunehmender Digitalisierung kontinuierlich. IT ist heutzutage nicht mehr nur ein Mittel zu Prozessoptimierung und Kosteneinsparung, sondern vielmehr ein Werkzeug, das das Potenzial hat, ganze Geschäftsmodelle radikal zu verändern. Aus diesem Grund haben auch IT-fremde Unternehmen großes Interesse an Digitalisierung. Der Gedanke ist folgender: Die erfolgreiche Digitalisierung eines Unternehmens baut auf Software. Wenn viele Unternehmen gleichzeitig diesem Ansatz folgen, wird der Arbeitsmarkt, was Entwicklerinnen und Entwickler angeht, »leergefegt«. Einerseits profitieren die Softwareentwicklungsunternehmen von dieser Marktsituation, d. h., die Auftragsbücher sind voll, und man kann für die Leistung einen höheren Preis verlangen. Andererseits stehen diese Softwareentwicklungsunternehmen vor einer großen Herausforderung: Es kann mitunter zum Problem werden, neue Mitarbeiterinnen und Mitarbeiter zu finden.

Für uns scheint diese Situation paradiesisch zu sein. Die Arbeitsplatzsicherheit ist hoch, ein Arbeitgeberwechsel ist selten riskant, und die Gehaltssituation ist auch zufriedenstellend. Allerdings muss man sich der zyklischen Entwicklung bewusst sein. In der Regel wird sich nach Phasen massiven Kapazitätsmangels auch wieder ein Überangebot einstellen. In diesem Fall stellt sich ein Gehaltsniveau als überhöht dar. Bei der Entscheidungsfindung sollte man sich die tatsächliche Dimension der Anforderungen vor Augen halten.

Gefragt sind neben fachlichen Fähigkeiten auch zunehmend sogenannte *Soft Skills*, zum Beispiel:

- Teamfähigkeit
- Aufgeschlossenheit
- Umsichtigkeit
- Kommunikation

Die Komplexität von Software erfordert es, dass man in der Softwareentwicklung heute im Team zusammenarbeitet und auch in der Lage sein muss, vernünftig zu kommunizieren. Vorbei sind die Zeiten, in denen man größtenteils allein mit unendlich vielen Tassen Kaffee zwischen schmutzigen Pizzapackungen am Quellcode »gewerkelt« hat.

Und was kann man damit so verdienen? Wie in anderen Bereichen auch ist die Spanne groß. Sowohl die geografische Region als auch die Spezialisierung der jeweiligen Person haben einen großen Einfluss. Abbildung 1.1 gibt einen Überblick über die durchschnittlichen Gehälter in unterschiedlichen IT-Bereichen innerhalb von Deutschland. Interessant: Programmiererinnen und Programmierer in den USA verdienen oft über 100.000 US-Dollar im Jahr (Quelle: *https://www.daxx.com/de/blog/entwicklungstrends/it-gehaelter-softwareentwickler-trends*.). Man sollte sich natürlich der Tatsache bewusst sein, dass sich Gehälter über Ländergrenzen hinweg nur schlecht miteinander vergleichen lassen; schon allein die Regelungen zu Sozialversicherung und Urlaub unterscheiden sich in den USA erheblich von denen in Deutschland.

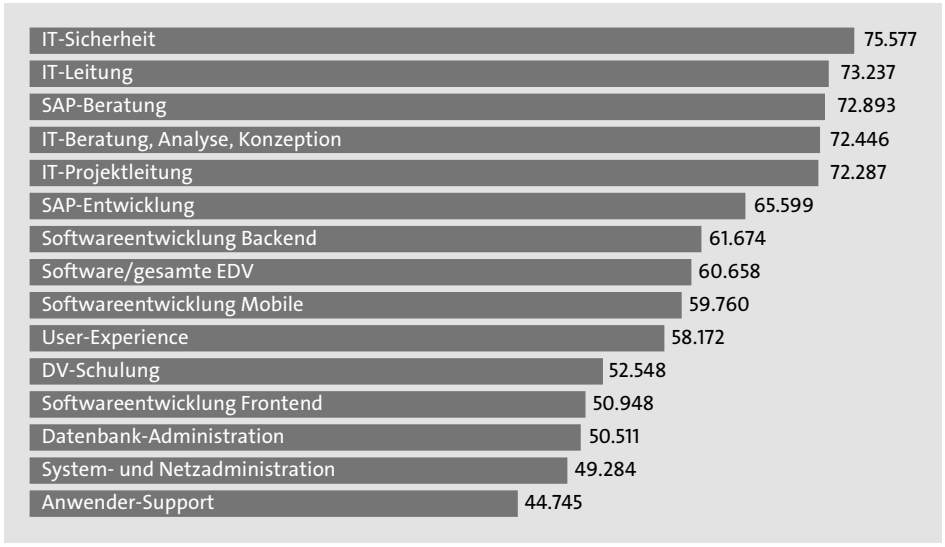

Abbildung 1.1 IT-Gehälter im Vergleich; Daten aus 2018 (Quelle: Statista)

Und wenn Sie jetzt noch immer zweifeln, ob dies der richtige Beruf für Sie ist, kann Ihnen vielleicht die Statistik in Abbildung 1.2 die Entscheidung ein wenig leichter machen. Die Statistik ist das Ergebnis einer Umfrage der bekannten Plattform *Stack*

Overflow. Stack Overflow unterstützt Sie, wenn Sie ein schwieriges Problem bei der Programmierung haben, auf dessen Lösung Sie nicht sofort kommen. Sie können Ihr Problem dort in einem Forum beschreiben, und andere Teilnehmer werden Ihnen gern antworten. Die »Kraft der Community« hat schon viele Probleme gelöst, an denen man allein fast verzweifelt wäre. Werfen Sie einen Blick auf die Seite der Plattform – es lohnt sich, versprochen.

Grundsätzlich ist es schwierig, den Zufriedenheitsgrad eines Menschen zu messen, doch es lassen sich einige Trends erkennen. In Deutschland scheint ein Großteil der Beschäftigten in der Softwareentwicklung mit ihrem Job im Großen und Ganzen zufrieden zu sein. Das liegt vermutlich auch daran, dass die inhaltlichen Herausforderungen, die die Erstellung von Software mit sich bringt, als ansprechend empfunden werden.

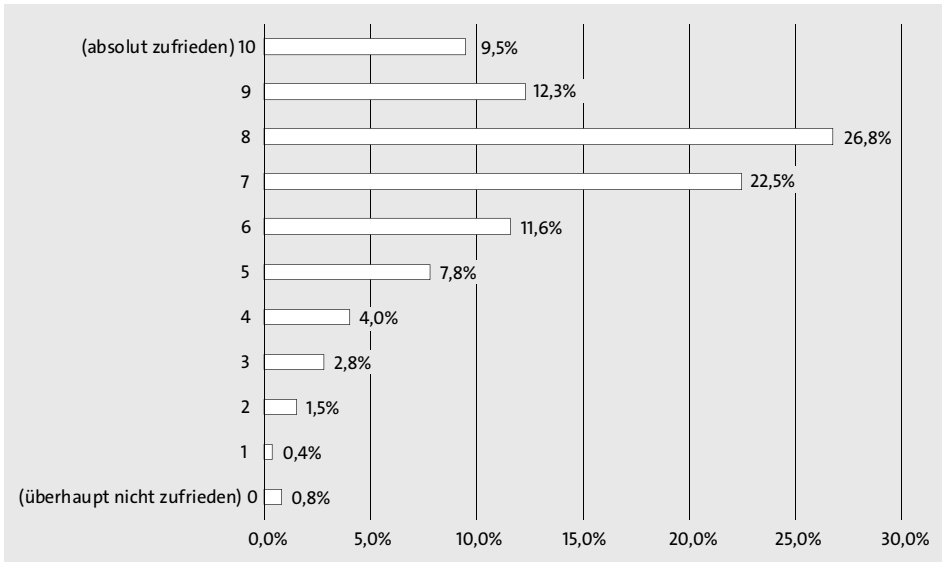

Abbildung 1.2 Zufriedenheit im Job bei Softwareentwicklern, Umfrage aus dem Jahr 2017 (Quelle: Stack Overflow, https://stackoverflow.com)

Dieses Buch möchte Sie auf dem Weg zur professionellen Softwareentwicklung begleiten. Seien Sie gespannt!

1.2 Über dieses Buch

Das Ziel des vorliegenden Handbuchs ist es, Ihnen einen umfassenden und dennoch kompakten Überblick über die vielfältigen Bereiche der Softwareentwicklung zu vermitteln. Das Themenspektrum der Softwareentwicklung ist heute so breit, dass es nicht gelingen kann, *alle* relevanten Inhalte in einem Buch wiederzugeben. Dennoch

ist es gerade am Anfang für einen angehenden Profi schwierig, sich selbst die richtigen Inhalte auszuwählen. Auf dem Markt gibt es unzählige Bücher, die dem Leser den Einstieg in die Programmierung mithilfe einer Programmiersprache näherbringen. Diese Art von Lehrbüchern beschäftigt sich jedoch oft nur mit der Programmierung der Software im engeren Sinn und vernachlässigt, dass bei einer professionellen Softwareentwicklung deutlich mehr Aufgaben zu bewältigen sind. Softwareentwicklung beginnt heute mit einer umfassenden Analyse der Anforderungen, die der Kunde an das neue Anwendungssystem stellt, und der Prozess der Entwicklung endet nicht mit dem Schreiben der letzten Zeile des Quellcodes. Das erstellte Anwendungssystem muss in der Systemumgebung Ihrer Kunden verfügbar gemacht und dort – gegebenenfalls in Zusammenarbeit mit dortigen Admins – integriert werden.

Aus eigenen Erfahrungen wissen wir, dass das Erstellen von Software in der Regel ein hochkomplexer Prozess ist, der intensiv geplant werden muss. Wir müssen uns in der Softwareentwicklung heute daher auch intensiv damit auseinandersetzen, auf welche Art und Weise der gesamte Entwicklungsprozess geplant, gesteuert und am Schluss realisiert wird. Dazu haben sich in der Vergangenheit unterschiedliche Vorgehensmodelle etabliert. Als professioneller Entwicklerin oder Entwickler müssen Sie die wichtigsten Vorgehensweisen und deren Vor- und Nachteile kennen.

Neben diesen Tätigkeiten, die direkt mit der eigentlichen Softwareentwicklung zu tun haben, erweitert sich unser Aufgabenspektrum immer mehr. Sie müssen heute zusätzlich etwas von der Gestaltung von Benutzeroberflächen und von Maßnahmen zur Qualitätssicherung verstehen. Da die IT in immer mehr Lebensbereiche vordringt, übernehmen Sie gewissermaßen Verantwortung über die zu verarbeitenden Daten. Sie müssen sich daher auch grundlegend mit den Regeln des Datenschutzes und der Datensicherheit auskennen.

Sie sehen, die fachlichen Anforderungen an Profis auf diesem Gebiet sind umfassend. Aus diesem Grund haben wir das vorliegende Handbuch in vier Teile gegliedert:

- Teil I: Überblick
- Teil II: Der Softwarelebenszyklus
- Teil III: Technologien und Methoden
- Teil IV: Trends

Sehen wir uns etwas genauer an, was Sie in den einzelnen Kapiteln erwartet.

Im ersten Teil des Buchs, »Überblick«, werden wir Ihnen die technischen Grundlagen der Softwareentwicklung vermitteln. Im vorliegenden ersten Kapitel haben Sie bereits etwas über das Berufsbild erfahren.

In Kapitel 2 mit der Überschrift »Programmierung als Kern der Softwareentwicklung« lernen Sie, was Programmierung eigentlich ist, wie ein Computerprogramm entsteht und was man unter den verschiedenen *Paradigmen* (Denk- und Herange-

hensweisen) der Softwareentwicklung versteht. Ebenso stellen wir Ihnen die Grundzüge der *objektorientierten Programmierung* vor. Das vorliegende Lehrbuch verwendet keine konkrete Programmiersprache, doch geben wir Ihnen einen Überblick über die Entwicklung von Programmiersprachen und stellen Ihnen einige der Sprachen, die heute häufig verwendet werden, detaillierter vor.

Technisch ins Detail geht es in Kapitel 3 unter dem Titel »Algorithmen und Datenstrukturen«. Algorithmen sind das Herz eines jeden Computerprogramms, denn sie beschreiben, in welcher Art und Weise die Daten verarbeitet werden. In diesem Kapitel stellen wir Ihnen einige wichtige Basisalgorithmen wie z. B. Suchverfahren vor. Jeder, der entwickelt, sollte umfassende Kenntnisse über diese Algorithmen besitzen. Das Thema Datenstrukturen ist gleichfalls wichtig, denn hier erfahren Sie die Basics zum Umgang mit Daten während der Programmerstellung. Nach dem Studium der Kapitel 2 und 3 haben Sie die wichtigsten Grundlagen erfahren, um zu verstehen, wie Computerprogramme auf technischer Ebene funktionieren.

Die Überschrift des zweiten Teils dieses Handbuchs, »Der Softwarelebenszyklus«, umreißt recht genau, um was es in diesem Teil des Buchs geht. Wir hatten bereits erwähnt, dass Softwareentwicklung mehr ist als das Schreiben von Quellcode.

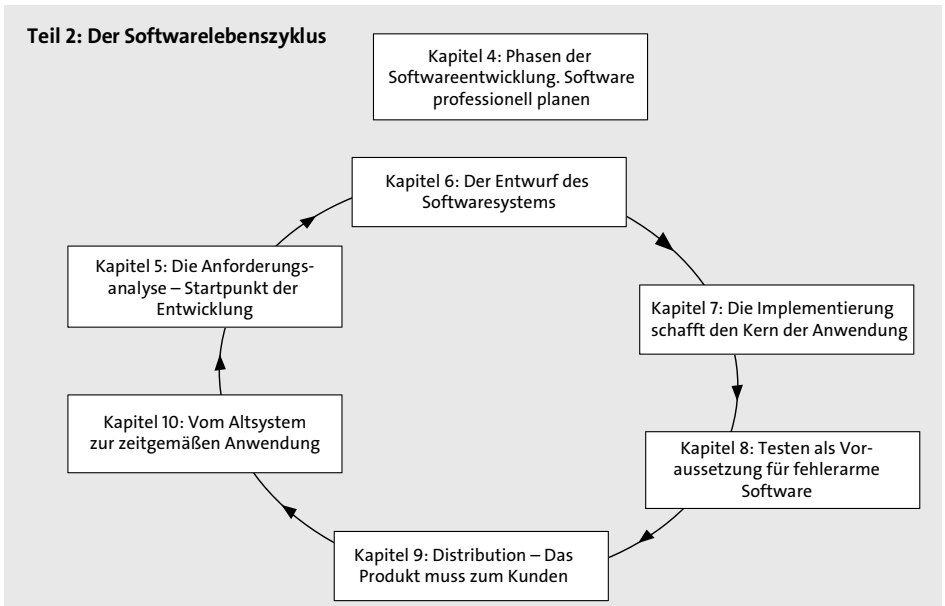

Abbildung 1.3 Kapitel in Teil II, Der Softwarelebenszyklus

Kapitel 4 unter der Überschrift »Softwareprojekte professionell planen« betrachtet den kompletten Planungsprozess eines Anwendungssystems. Dabei stellen wir Ihnen die unterschiedlichen Vorgehensmodelle vom Wasserfallmodell bis hin zu den agilen Methoden vor. In den folgenden Kapiteln tauchen wir genauer in die ein-

zelnen Phasen des Entwicklungsprozesses ein. Kapitel 5, »Die Anforderungsanalyse – Startpunkt der Entwicklung«, beschäftigt sich mit den vielen Aspekten rund um das Thema Anforderungen. Hier geht es darum, zu ergründen, welche Ansprüche unsere Kunden an die zu erstellende Software haben und wie wir diese Anforderungen ermitteln, sortieren und in den Arbeitsprozess einfließen lassen können.

Nach der Ermittlung der Anforderungen geht man in der Regel dazu über, das zu erstellende Anwendungssystem zu konzipieren. Typische Aufgaben dieses Schritts sind die Festlegung der Architektur um ein solides Fundament für die Software sowie das Anfertigen von Prototypen. Diese und ähnliche Fragen stehen im Mittelpunkt von Kapitel 6, »Der Entwurf des Softwaresystems«.

Der Kern der Entwicklung passiert in der Phase der Implementierung. Die Fragen rund um dieses Thema sind Gegenstand von Kapitel 7, »Die Implementierung schafft den Kern der Anwendung«. Dabei werden wir uns die typischen Ebenen eines Anwendungssystems, d. h. die Benutzeroberfläche, die *Businesslogik* und die Schicht zur Anbindung der Datenhaltung genauer ansehen.

Nach der Implementierung liegen bestimmte Teile des Anwendungssystems in einem lauffähigen Zustand vor, doch der Lebenszyklus ist noch nicht am Ende. Der nächste logische Schritt ist das Testen der Software. Möchten Sie mehr über *Unit-*, *Benutzer-* oder spezielle *Betatests* erfahren, hilft Ihnen Kapitel 8, »Testen als Voraussetzung für fehlerarme Software«, hoffentlich weiter.

Auch mit der leistungsfähigsten Software können wir nur dann Geld verdienen, wenn wir sie unserer Kundschaft zur Verfügung stellen. Die Aufgaben der Softwaredistribution sind vielfältig. Über einige wichtige Aspekte, zum Beispiel über die Möglichkeiten der technischen Bereitstellung oder über die Bedeutung der heutigen App-Stores, lesen Sie in Kapitel 9, »Distribution – das Produkt muss zum Kunden«.

Mit Kapitel 10, »Vom Altsystem zur zeitgemäßen Anwendung«, schließt sich gewissermaßen der Kreis des Softwarelebenszyklus. Auch Software altert! Thema dieses Kapitels ist die sogenannte *Softwaremigration*. Wir stellen Ihnen Wege und Methoden vor, mit denen Sie Altsysteme wieder fit für die Zukunft machen können. Das Ziel dabei ist es, so viel wie möglich zu erhalten, und zwar besonders dort, wo eine komplette Neuentwicklung zu aufwendig und zu teuer wäre.

Im dritten Teil des Handbuchs, »Technologien und Methoden«, haben wir Ihnen eine Auswahl an Themen zusammengestellt, die sich vertiefend mit den vielfältigen Facetten des Entwicklungsprozesses auseinandersetzen.

Moderne Software wird heute sehr oft in Form von Webapplikationen erstellt. In Kapitel 11, »Webtechnologien«, erfahren Sie die Basics der Webprogrammierung. Hier stehen Themen wie HTML, CSS und JavaScript auf der Agenda. Damit stellen wir sicher, dass Sie die Grundlagen der modernen Webprogrammierung beherrschen, denn ohne diese Basics werden Sie auch unter Einsatz eines noch so großartigen Werkzeugs oder einer leistungsfähigen Bibliothek nicht zum Ziel kommen.

Ein schon lang anhaltender Trend sind Apps für die mobilen Systeme. Wie man diese programmiert, welche Ansätze es gibt und welche Vorteile eine native App gegenüber einer hybriden App hat, das alles lesen Sie in unserem Kapitel 12, »Apps für mobile Systeme«. Sie dürfen schon mal vorausblicken, aber kommen Sie bitte wieder an diese Stelle zurück. Wir haben noch weitere spannende Themen für Sie vorbereitet.

Unsere Kundschaft nutzt im Büro den Desktopcomputer, zu Hause lesen sie die Informationen auf dem Tablet, und unterwegs werden Apps auf dem Smartphone verwendet. Auch bei den Betriebssystemen ist die Vielfalt groß: Microsoft Windows, macOS und Linux auf der einen Seite sowie Android und iOS auf der anderen. Bei der Softwareentwicklung ist es eine Herausforderung, die Software für möglichst alle diese Systeme und Geräte tauglich zu machen. Wie aus heutiger Sicht der Stand der Dinge zu diesem Thema ist, das haben wir für Sie in Kapitel 13, »Plattform- und geräteübergreifende Entwicklung«, zusammengestellt.

»Parallel geht schnell«, das gilt auch für Computerprogramme – jedoch nur dann, wenn Ihre Software in der Lage ist, die Voraussetzungen moderner Computertechnologie mit Mehrkernprozessoren auch zu nutzen. Lesen Sie dazu bitte Kapitel 14, »Parallelprogrammierung«.

Software soll heutzutage nicht nur möglichst fehlerfrei funktionieren, sondern den Menschen auch nutzen, sie bei ihrer Arbeit unterstützen und im besten Fall Spaß machen. Was versteht man unter Softwareergonomie? Wie soll ein benutzerfreundliches Interface gestaltet sein, und gibt es dafür Richtlinien und Vorgaben? Lesen Sie zu diesen Fragen Kapitel 15, »Kundenzufriedenheit durch Nutzerorientierung«.

Software durchdringt sowohl privat als auch beruflich immer mehr Lebensbereiche. Als Folge davon werden durch unsere Programme immer mehr und vor allem auch immer sensiblere Daten verarbeitet. Wir können nicht so tun, als ginge uns das als Entwicklerinnen und Entwickler nichts an! Kapitel 16, »Datensicherheit und Datenschutz«, beschäftigt sich genau damit.

Nahezu jedes Computerprogramm greift auf Daten zurück. Grundlegende Datenbankkenntnisse sind für Sie als Entwickler unverzichtbar. Stichwörter aus Kapitel 17, »Grundlagen der Datenhaltung«, sind *relationale Datenbank*, *Normalisierung* und *ER-Modell*.

Was der Hammer für einen Handwerker ist, das ist die Entwicklungsumgebung für uns Entwickler! Es gibt eine unfassbar große Auswahl an Softwaretools, die einen bei allen Aufgaben des Entwicklungsprozesses unterstützen können. Einen Überblick über das Thema erhalten Sie in Kapitel 18 dieses Handbuchs unter der Überschrift »Werkzeugunterstützung«.

Alle reden von Qualität, auch im Zusammenhang mit Software. Aber was ist das im Fall von Software eigentlich genau? Einen Einstieg in dieses immer wichtiger werdende Thema gibt Ihnen Kapitel 19, »Qualitätssicherung und Clean Code Development«.

Haben Sie die Teile 1 bis 3 des Handbuchs durchgearbeitet, sind Sie bereits sehr umfassend über die Softwareentwicklung als Ganzes informiert worden. Gibt es da noch mehr? Ja, noch viel mehr! So viel, dass es auch nicht in zehn solcher Handbücher passen würde. Dennoch wollten wir noch nicht aufhören und haben einen vierten Teil namens »Trends« erstellt. Viele allgemeine Entwicklungen der IT beeinflussen auch Fragen, Aufgaben sowie Art und Weise der Softwareentwicklung.

Das Thema »Enterprise Mobile Computing« greifen wir in Kapitel 20 auf. Es beschäftigt sich mit Aufgaben und Fragen rund um die Nutzung mobiler Computertechnologie in Unternehmen. Diese unterscheidet sich essenziell von der privaten Nutzung von Smartphone und Tablet. Es kommen neue und spannende Aufgaben auf uns zu.

Vom Kühlschrank bis zum selbstfahrenden Auto soll heute alles vernetzt und über das Internet angebunden werden. Das *Internet der Dinge* ist inzwischen nicht mehr nur ein Hype, sondern es kommt nach und nach auch in der Praxis an. Und was geht uns das an? Eine ganze Menge, denn wir sind diejenigen, die die dazu notwendige Software programmieren sollen. In Kapitel 21, »Internet of Things«, gehen wir dem Thema auf den Grund.

Das Beste kommt bekanntermaßen zum Schluss. Zumindest ist das Thema von Kapitel 22, »Cloud-Computing«, hochaktuell, und es verändert die IT und unsere Arbeitsweise zunehmend. Nach einer Einführung in die Grundlagen des Cloud-Computings sehen wir uns genauer an, welche Auswirkungen schon heute für die Softwareentwicklung relevant sind.

Das Buch ist Lehrbuch und Nachschlagewerk zugleich. Je nachdem, wie Ihr aktueller Wissenstand ist und welche Frage Sie gerade dazu motiviert, sich in der Literatur auf die Suche zu begeben, können Sie das vorliegende Buch von vorn nach hinten durcharbeiten oder auch nur punktuell nach Bedarf einzelne Kapitel studieren. In jedem Kapitel haben wir Ihnen wichtige Fachbegriffe hervorgehoben, und jedes Kapitel endet mit einer Zusammenstellung von gedruckten Quellen und Internetlinks.

Viel Spaß beim Lesen und Lernen wünscht das Autorenteam!

1.3 Quellen der zitierten Statistiken

https://www.uni-oldenburg.de/informatik/studium-lehre/studium-und-beruf/berufsbild-informatik

http://www.karista.de/berufe/softwareentwickler

https://www.daxx.com/de/blog/entwicklungstrends/it-gehaelter-softwareentwickler-trends

https://stackoverflow.com/

Kapitel 8

Testen als Voraussetzung für fehlerarme Software

Fehlerfreie Software ist und wird wohl auch künftig ein Traum bleiben. Zu komplex und umfassend sind Anwendungen, als dass man alle Einsatzszenarien und Umfeldbedingungen antizipieren könnte. Um fehlerarme Software zu erstellen, sind Tests notwendig.

Das Thema »Softwaretests« ist äußerst umfangreich. Ganz traditionell hat man die Tests immer am Ende eines Entwicklungszyklus eingeordnet. Im Rahmen des Wasserfallmodells (Abbildung 8.1) ist es die letzte Phase vor dem aktiven Betrieb der Software in der Arbeitsumgebung unserer Kundinnen und Kunden.

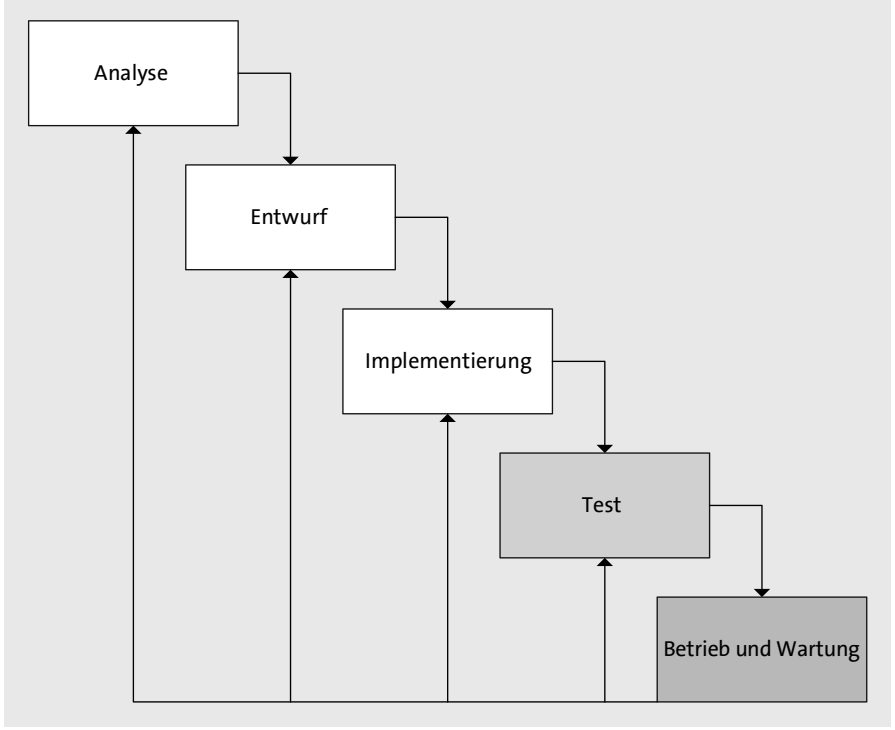

Abbildung 8.1 Die Testphase im Rahmen des Wasserfallmodels (Quelle: https://de.wikipedia.org/wiki/Wasserfallmodell)

Diese Sichtweise ist aus heutiger Perspektive natürlich nicht mehr haltbar. Viel früher im Entwicklungszyklus, fortlaufend und auf nahezu allen Ebenen muss Software getestet werden. Die Bedeutung von Softwaretests kann man unter anderem auch daraus ableiten, dass man die Reihenfolge von Entwicklung und Tests bei der sogenannten *testgetriebenen Entwicklung* (*Test-Driven Development*, *TDD*) zugunsten der Softwaretests umdreht. Wir widmen uns diesem wichtigen Ansatz des Testens von Software in Abschnitt 8.2, »Testgetriebene Entwicklung«. Sehr wichtig ist es, die verschiedenen Arten von Softwaretests zu unterscheiden. Das Spektrum reicht von einer Überprüfung der Funktionen eines Anwendungssystems bis hin zur Feststellung, wie gut sich die Software an künftige Anforderungen anpassen lässt. Die Testarten sind Gegenstand von Abschnitt 8.3, »Ein Überblick über wichtige Testarten«. Die Art und Weise des Testens sagt etwas darüber aus, zu welchem Zeitpunkt der Test erfolgt. Einige Tests kann man zur Entwicklungszeit durchführen, andere Tests erst, wenn die Anwendung gestartet ist. Beide Testarten werden in Abschnitt 8.4, »Testmethoden«, vorgestellt.

Wir hatten es bereits erwähnt: Man kann die unterschiedlichsten Aspekte einer Software auf ihre Tauglichkeit hin untersuchen. Wir sprechen in diesem Zusammenhang von Testebenen. Dies ist das Thema von Abschnitt 8.5, »Testebenen«. Wichtig schon an dieser Stelle: Tests passieren nicht immer auf technischer Ebene. Zunehmend wichtiger wird es, zu prüfen, ob alle Ansprüche der künftigen Nutzer erfüllt werden. Nutzertests liefern damit gewissermaßen einen finalen Eindruck der Software und versuchen die Frage aller Fragen zu beantworten: Ist die Anwendung so, wie es sich alle Beteiligten vorgestellt haben? Diese Frage beinhaltet sämtliche Facetten – vom Fehler in der Programmlogik über die Reaktionsfähigkeit der Software bis hin zum Gefallen der Benutzeroberfläche.

Manche Tests muss man nach wie vor manuell ausführen. Andere Tests müssen automatisiert werden, um sie oft, regelmäßig und umfassend durchführen zu können. Die Technik des Testens, also den Einsatz von typischen Tools, und die Testautomatisierung mit ihrer Einbindung in den Build-Prozess beschreibt Abschnitt 8.6, »Technik des Testens«. Beginnen wir jedoch damit, ein paar wenige Grundbegriffe voneinander abzugrenzen und zu klären, wofür Softwaretests eigentlich da sind.

8.1 Zur Notwendigkeit von Softwaretests

Das Ziel von Softwaretests ist es, die Qualität von Programmen zu erhöhen. Gelingt das, wird dadurch das Vertrauen der User in die Software erhöht. Um das zu erreichen, ist es das Ziel, möglichst viele Fehler in der Phase der Entwicklung zu finden, also bevor die Software bei den Kunden eingesetzt wird.

Im Bereich der Softwaretests kann man zwischen *Validierung* und *Verifikation* unterscheiden (Abbildung 8.2).

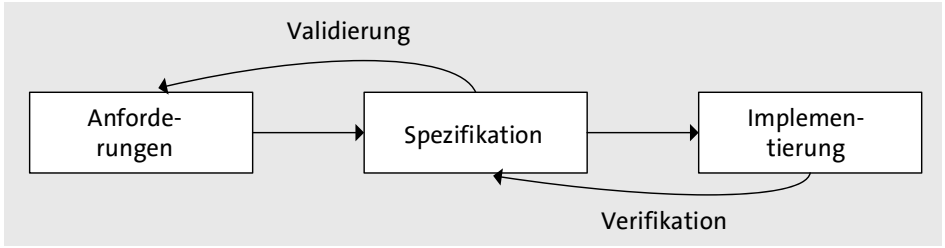

Abbildung 8.2 Validierung und Verifikation (Quelle: Ehmke, 2011)

Wo liegen die Unterschiede? Mit der Validierung überprüft man, ob die erstellte Softwarespezifikation, die als Grundlage für die folgende Implementierung dient, korrekt ist. Mit anderen Worten: Es wird dabei geprüft, ob das Entwicklungsteam seine Kunden und seine Zielgruppe richtig verstanden hat. Probleme und Fehler können dabei unabhängig vom Entwicklungsansatz (Wasserfall oder agil) entstehen. Der Schritt der Verifikation umfasst die Überprüfung der Implementierung, d. h., ob die Umsetzung in der gewählten Programmiersprache korrekt erfolgt ist.

Softwaretests verfolgen also das Ziel, die Eigenschaften eines Anwendungssystems zu ermitteln und dabei die Unterschiede zwischen dem tatsächlichen Zustand (Ist) und dem erforderlichen Zustand (Soll) festzustellen.

8.2 Testgetriebene Entwicklung

Wie bereits am Anfang dieses Kapitels dargestellt, wird bei der testgetriebenen Entwicklung (*Test-Driven Development*, *TDD*) bezüglich der Reihenfolge von Entwicklung und Test in entgegengesetzter Weise vorgegangen. Hier werden die Tests dazu genutzt, die Softwareentwicklung zu steuern. Den Ablauf der Programmierung können Sie sich daher wie in Abbildung 8.3 gezeigt vorstellen.

Es werden drei Zyklen durchlaufen:

1. Ein Test wird geschrieben, der zunächst fehlschlägt (Rot).
2. Gerade so viel *Produktivcode* wird implementiert, dass der Test erfolgreich durchläuft (Grün).
3. Test und Produktivcode werden refaktorisiert, d. h. überarbeitet.

Die testgetriebene Entwicklung ist inkrementell, d. h., jeder Zyklus erweitert die Software um neue Fähigkeiten. Dabei soll sehr kleinteilig vorgegangen werden: Ein Zyklus sollte nur ein paar Minuten dauern.

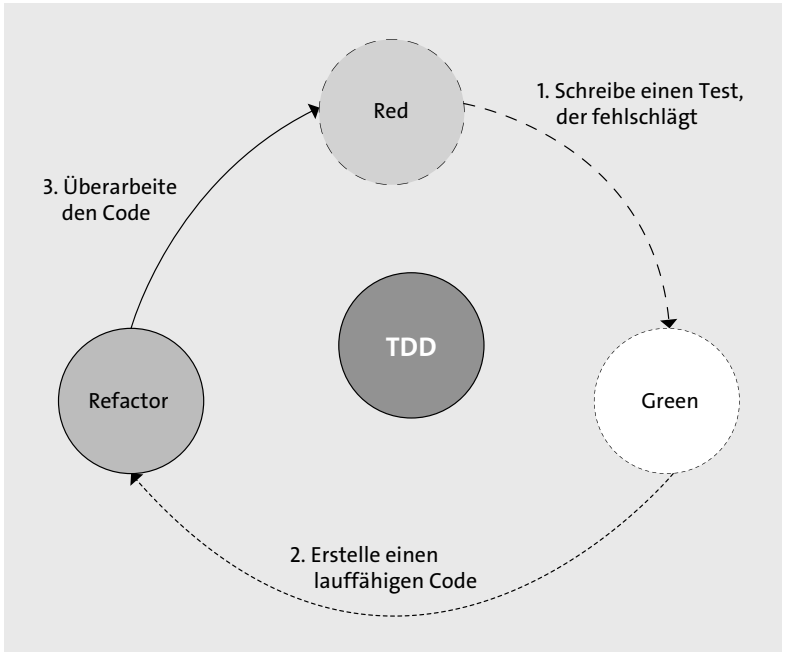

Abbildung 8.3 Ablauf der testgetriebenen Softwareentwicklung (in Anlehnung an https://www.it-agile.de/wissen/agiles-engineering/testgetriebene-entwicklung-tdd)

Das Besondere an dieser Vorgehensweise ist, dass die Tests geschrieben werden, bevor die Komponente implementiert wird. Man bezeichnet dieses Vorgehen auch als *Test-First*. Das Entwicklungsteam bekommt während der Entwicklung direktes Feedback. Tests und Produktivcode können durchaus von unterschiedlichen Personen geschrieben werden. Damit vermeidet man den eigenen Interessenkonflikt, schreibt einen Test, ohne auf die Implementierung zu achten, und produziert nur so viel Produktivcode, wie wirklich notwendig ist. Beim Refactoring werden die Tests und der Produktivcode gleichermaßen aufgeräumt. Ziel hierbei ist es, die Software einfach, redundanzfrei und verständlich zu gestalten.

TDD ist ein Kernelement der agilen Softwareentwicklung und soll einen großen Beitrag zu einer qualitativ hochwertigeren Software liefern. Die Vorzüge dieses Vorgehens liegen unmittelbar auf der Hand:

- kein ungetesteter Code
- eine saubere und testbare Architektur durch TDD als Designstrategie
- keine bzw. wenige Redundanzen durch stetes und rechtzeitiges Refaktorisieren
- kein unnötiger Produktivcode (es wird nur immer so viel Code generiert, wie zur Erfüllung des Tests notwendig ist)
- Konzentration auf den Kern

Gelegentlich hören wir, dass die testgetriebene Softwareentwicklung insgesamt zu aufwendig ist. Der Grund ist, dass gewissermaßen jedes Problem zweimal angefasst werden muss: einmal im Test und einmal beim Schreiben des Produktivcodes. Dieses Argument lässt sich jedoch sehr schnell entkräften, da Software auf jeden Fall nahezu »flächendeckend« getestet werden sollte und bei dieser Vorgehensweise gerade Einsparungen bei unnötigem Code erfolgen. Das Hauptziel bleibt jedoch die größere Fehlerfreiheit und damit eine höhere Softwarequalität.

8.3 Ein Überblick über wichtige Testarten

In diesem Kapitel betrachten wir die unterschiedlichen Testarten. Wir können beispielsweise folgende Einteilung vornehmen:

- funktionale Tests
- nicht funktionale Tests
- strukturbezogene Tests
- änderungsbezogene Tests

Sehen wir uns das im Einzelnen an. Bei einem *funktionalen Test* wird das System auf sein Ein- und Ausgabeverhalten geprüft. Basis für die Bewertung sind die sogenannten funktionalen Anforderungen, die sich unter anderem aus den Dokumenten der Anforderungsanalyse, wie Lasten- und Pflichtenheft, oder aus dem Backlog ergeben. Meist werden diese Arten von Tests als Black-Box-Verfahren durchgeführt. Eine funktionale Anforderung wird in der Regel nicht durch einen einzigen Testfall, sondern durch das Zusammenspiel mehrerer Testfälle überprüft.

Bei *nicht funktionalen Tests* geht es im Wesentlichen darum, wie gut das untersuchte System funktioniert. Typische nicht funktionale Tests sind in Tabelle 8.1 aufgeführt.

Nicht funktionaler Test	Beschreibung
Lasttest	Hier wird das Systemverhalten bei steigender Benutzer- oder Transaktionszahl gemessen. Dabei wird die Last schrittweise bis zu dem Punkt erhöht, den die Spezifikation vorsieht.
Performancetest	Wir messen die Verarbeitungs- und Antwortzeiten des Systems. Diese sind meist abhängig von der Last, sodass die Werte gerade bei hoher Last geprüft werden müssen.
Volumentest	Wie verhält sich das System bei der Bewältigung sehr großer Datenmengen?

Tabelle 8.1 Arten nicht funktionaler Tests

Nicht funktionaler Test	Beschreibung
Stresstest	Die durch das System zu verarbeitende Last (Benutzer-/Transaktionszahl) wird über die festgelegten Grenzwerte hinaus gesteigert. Das Ziel: Prüfung des Systems bei Überlastung.
Security-Test	Das Ziel ist es, die Sicherheit des Systems vor unberechtigten Datenzugriffen zu prüfen. Gelingt es von außen, ohne das Vorhandensein von Berechtigungen in das System einzudringen?
Zuverlässigkeitstest	Wie reagiert das System unter Dauerbetrieb? Notiert wird beispielsweise die Anzahl der Ausfälle in einem bestimmten Zeitintervall.
Robustheitstest	Die Robustheit beschreibt, wie resistent das Programm gegenüber externen Fehlern ist, zum Beispiel bei Fehleingaben durch Nutzer.
Kompatibilitätstest	Wie leicht lassen sich die Daten der Software mit anderen Programmen austauschen? Gibt es hier eine entsprechende Kompatibilität bezüglich des verwendeten Datenformats, oder sind leistungsfähige Import- und Exportfilter vorhanden?
Konfigurationstest	Wie gut lässt sich das System in unterschiedlichen Systemumgebungen verwenden? Dies betrifft zum Beispiel unterschiedliche Betriebssysteme, Versionen von Betriebssystemen, Hardware oder Landessprachen.
Usability-Test	Kann das System als benutzerfreundlich gelten? Auf diesen Aspekt kommen wir in Abschnitt 8.5.4, »Abnahme- und Nutzertests«, nochmals zurück.
Dokumentationstest	Zu Software gehört auch sehr oft eine Dokumentation. Zum einen umfasst diese eine Nutzerdokumentation, zum Beispiel in Form einer integrierten Onlinehilfe, zum anderen die Softwareentwicklungsdokumentation. Beide Dokumente müssen vollständig und korrekt sein, nur dann sind sie eine sinnvolle Hilfe.

Tabelle 8.1 Arten nicht funktionaler Tests (Forts.)

Kommen wir zu den sogenannten *strukturbezogenen Tests*. Das Ziel ist es, die interne Struktur des betreffenden Softwareelements zu durchleuchten. Diese Tests erfolgen als White-Box-Tests. Dabei soll der Kontroll- und Datenfluss des Objekts überprüft

werden. Idealerweise wird eine vollständige Testabdeckung erreicht, d. h., alle Strukturen und Verzweigungen werden mindestens einmal durchlaufen und überprüft.

Da Software in der Regel ständig an neue Anforderungen angepasst werden muss bzw. da Updates bereitgestellt werden, die bekannte Fehler beseitigen sollen, muss auch immer wieder neu getestet werden. Als *änderungsbezogene Tests* bzw. *Regressionstests* bezeichnet man Softwaretests, die nach einer Modifikation des Anwendungssystems durchgeführt werden. Dabei soll sichergestellt werden, dass durch die Änderungen keine neuen Fehler in die Anwendung hineingekommen sind. Dies bezeichnet man als unerwünschte Seiteneffekte. Die Tests umfassen sowohl funktionale als auch nicht funktionale Aspekte. Je öfter Änderungen des Softwaresystems stattfinden, desto umfangreicher ist auch der Bedarf an solchen Tests. Um die Menge der Tests handhaben zu können, ist das Ziel eine weitgehende Testautomatisierung. Ein Regressionstest kann dabei folgende Teilbereiche umfassen:

- Test der Funktionen, die Anlass für die Programmrevision waren
- Test aller Programmfunktionen, die durch die Programmrevision angepasst wurden
- Test aller Programmteile, die neu in das Programm aufgenommen wurden
- Test des kompletten Systems, d. h. ein vollständiger Regressionstest

Nicht nach jeder Programmänderung kann ein vollständiger Testzyklus des gesamten Programmsystems durchgeführt werden. Dies wäre zu umfangreich. Der ideale Testumfang resultiert hier aus einer Abwägung zwischen dem Aufwand für einen Test und dem verbleibenden Risiko, wenn bestimmte Tests nicht ausgeführt werden.

Die Situation in der Praxis können Sie sich gut anhand von Abbildung 8.4 verdeutlichen.

Dazu sind ein paar Erläuterungen angebracht: Auf der x-Achse sehen Sie den Zeitverlauf. Auf der y-Achse können Sie den Anteil an getestetem und ungetestetem Code ablesen. Wir befinden uns am Punkt der vertikalen Linie mit der Bezeichnung »beginne Tests«. Ab diesem Zeitpunkt finden auch Anpassungen an der Software statt. Der Quellcode besteht zu jedem Zeitpunkt aus einem bestimmten Anteil an ungetestetem Code. Wie groß dieser Anteil ist, ist nicht immer bekannt. Das ist zum Beispiel der Fall, wenn Sie ein bestehendes Projekt weiterentwickeln sollen und dies nicht aus der Dokumentation hervorgeht. Änderungen am Softwareprojekt, d. h. am Quellcode, finden auf zwei Ebenen statt. Zum einen kann das Projekt erweitert werden, indem neuer Programmcode hinzugefügt wird. Dieser neue Programmcode wird natürlich bestmöglich getestet, sodass sich der Anteil von ungetestetem Code, relativ betrachtet, verringert. Zum anderen werden bestimmte Programmfunktionen auch angepasst bzw. geändert. Diese Programmänderungen werden ebenso getestet.

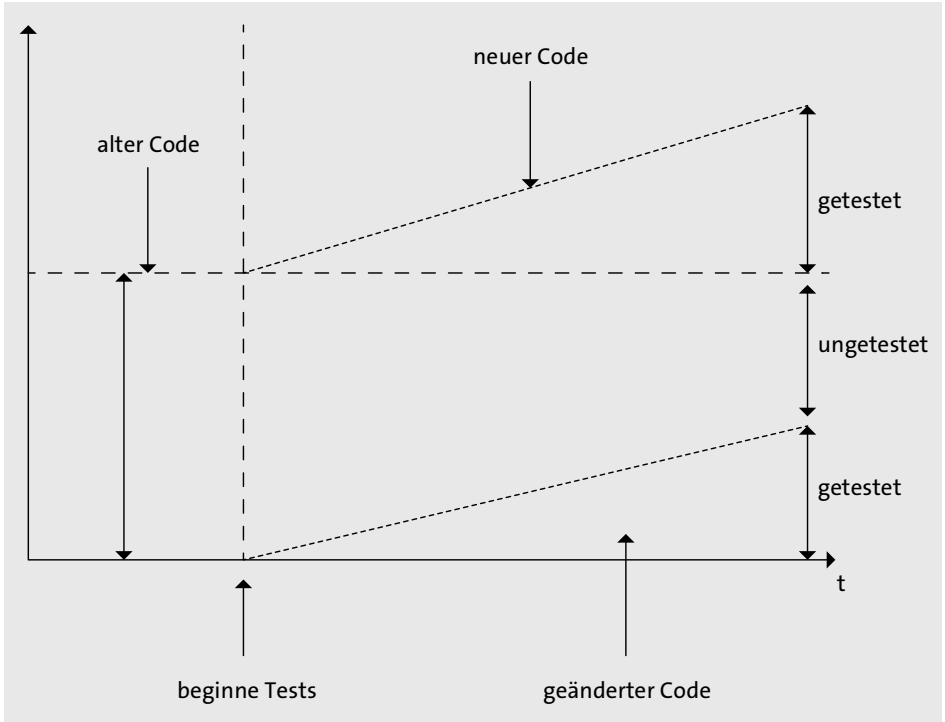

Abbildung 8.4 Regressionstests führen zu einer Erhöhung der Testabdeckung durch Testen von neuem und geändertem Code. (Quelle: https://www.johner-institut.de/blog/iec-62304-medizinische-software/regressionstest)

Dieses Vorgehen ist praxisrelevant, da man auf diese Weise bestehende Programme erweitern bzw. anpassen kann und gleichzeitig den Abdeckungsgrad des Quelltextes durch Tests sukzessive erhöht. Altanwendungen können auf diese Weise bei notwendigen Anpassungen auch durch Tests in der Qualität erhöht und damit für den weiteren Betrieb bzw. für weitere Pflege fit gemacht werden. Die Aufwendungen für die Tests können wir dann gegenüber unseren Auftraggebern auch leichter rechtfertigen.

Best Practices der Regressionstests

Bei der Durchführung von Regressionstests können Sie sich die Arbeit erleichtern, wenn Sie die folgenden Tipps berücksichtigen:

- *Automatisierung*: Die Automatisierung von Tests ist die beste Investition in die Effizienz. Zu Beginn mag diese etwas mehr Aufwand machen, aber es zahlt sich aus. Bei jeder Programmänderung ist der Code, der durch automatisierte Tests abgedeckt wird, in kürzester Zeit überprüft.

- *Sukzessive Erhöhung der Testabdeckung*: Wahrscheinlich werden Sie für bestehenden Code keine Tests mehr schreiben wollen. Gehen Sie bei Programmanpassungen und -erweiterungen jedoch so vor, wie dies anhand von Abbildung 8.4 beschrieben wurde: Erhöhen Sie im Laufe der Zeit den Anteil an getestetem Code.
- *Investitionen in die Softwarearchitektur*: Je besser die Struktur Ihrer Anwendung ist, desto leichter lassen sich auch Tests durchführen. Verbessern Sie daher nach Möglichkeit die Architektur der Software. Eine weitgehende Entkopplung der Schichten sorgt zum Beispiel dafür, dass sich die einzelnen Programmebenen besser unabhängig testen lassen.
- *Häufig testen*: Sie sollten die Tests möglichst häufig angestoßen. Das geht umso leichter, je größer der Grad der Automatisierung ist. Hilfreich ist beispielsweise die Verwendung eines Build-Tools. Bevor Software in das zentrale Repository übertragen werden kann, muss sie einen umfassenden Testkanon durchlaufen.

So, jetzt wissen Sie, welche Arten von Tests es grundsätzlich gibt. Oder, mit anderen Worten: Sie wissen, was man alles bei einem Anwendungssystem testen kann. Ebenso wichtig ist Frage nach der Vorgehensweise. Mit den wichtigsten Testmethoden beschäftigen wir uns jetzt.

8.4 Testmethoden

Beim Testen unterscheidet man zwischen zwei grundsätzlichen Testmethoden. Zum einen sind dies die *statischen Tests*, die durchgeführt werden, ohne dass die Software ausgeführt wird. Dagegen werden *dynamische Tests* erst zur Laufzeit der Anwendung durchgeführt. Sie machen den Kern eines Softwaretests aus. Beide Testmethoden stellen wir jetzt vor.

8.4.1 Statische Tests

Unter einem statischen Test versteht man die Analyse bzw. die Überprüfung des Quellcodes des betreffenden Testobjekts, ohne dass dieser ausgeführt wird. Zwei größere Untergruppen können wir hier unterscheiden:

1. werkzeugunterstützte Analyse
2. strukturierte Gruppenprüfungen, auch als Reviews bezeichnet

Sehen wir uns das wieder etwas genauer an. Bei der werkzeugunterstützten Analyse erfolgt zur Entwicklungszeit eine Überprüfung des Quellcodes auf bestimmte Parameter, zum Beispiel auf unerreichbare Codebestandteile, Verstöße gegen Codierrichtlinien, nicht verwendete Variablen und Objekte usw. Dazu stehen uns umfassende Tools zur Verfügung, die teilweise schon in die Entwicklungsumgebungen integriert

wurden bzw. über Plug-ins nachgerüstet werden können. Den Umfang der Prüfungen können Sie dabei auch konfigurieren. Das Thema greifen wir in Abschnitt 19.4.3, »Regeln als Handlungsleitfaden«, wieder auf, wenn es um Maßnahmen zur Sicherung der Qualität bei der Softwareentwicklung geht.

Kommen wir zu den strukturierten Gruppenprüfungen. Diese Form von Tests basieren weitgehend auf den menschlichen Analysefähigkeiten und können damit nur bedingt durch Tools unterstützt werden. Im Grunde geht es darum, dass Programmstücke bzw. Dokumente durch eine einzelne Person oder durch mehrere Personen geprüft werden. Man unterscheidet verschiedene Arten von strukturierten Gruppenprüfungen anhand ihres Grads der Formalisierung (Abbildung 8.5).

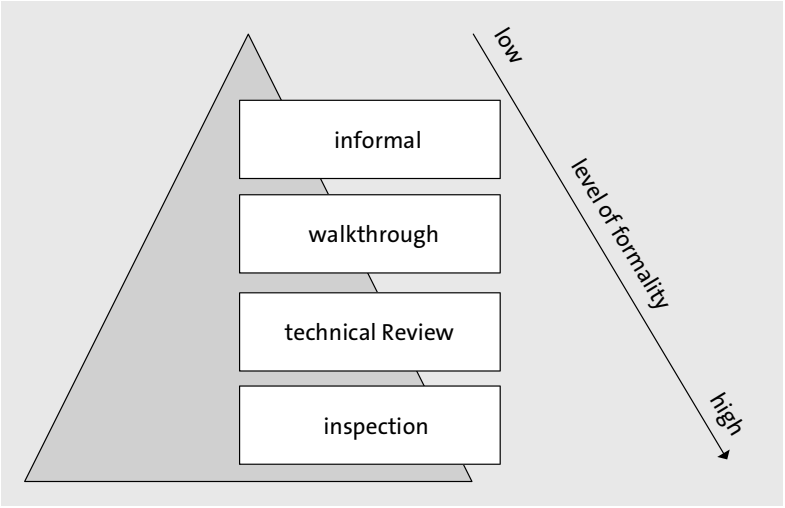

Abbildung 8.5 Formen der strukturierten Gruppenprüfungen (Quelle: https://www.tutorialspoint.com/software_testing_dictionary/technical_review.htm)

Worin bestehen die Unterschiede? Das versuchen wir, gleich zu erklären.

- *Informal*: Es handelt sich um eine informelle Review, die auf keinen festgelegten Regeln basiert. Der Ablauf der Prüfung des Quellcodes erfolgt weitgehend nach der Intuition des Prüfers oder der Prüferin.
- *Walkthrough*: Hier stellt der Autor des Quellcodes seine Arbeit in einem Meeting Schritt für Schritt vor. Die Gutachterin hört aufmerksam zu und versucht dabei, mögliche Fehler und Schwachstellen zu entdecken. Auch diese Vorgehensweise ist nur wenig formal, denn die Inhalte der Diskussion ergeben sich ad hoc während der Präsentation. Der Vorteil besteht zum einen darin, dass der Autor selbst zu einer Überprüfung gezwungen ist, wenn er seine Arbeit vorstellt, und zum anderen, dass weitere Personen versuchen, die Arbeitsschritte nachzuvollziehen. Üblicherweise bittet die Programmautoren zum Walkthrough und steuert diesen auch.

- *Technische Review*: Bei einer Review handelt es sich um einen dokumentierten und definierten Fehlerfindungsprozess. Die Überprüfung des Quellcodes bzw. des Dokuments erfolgt auf der Basis vorher definierter Prüfschritte und -kriterien. Sie verwenden dazu zum Beispiel Checklisten. Die Ziele einer solchen Review bestehen darin, Fehler zu finden, Alternativen zu eruieren, über Lösungsoptionen zu diskutieren und die Einhaltung von Spezifikationen zu prüfen. Die technische Review wird durch eine Moderatorin oder einen Moderator geleitet und nicht durch die Autoren.
- *Inspektion*: Es ist die formalste Variante für eine strukturierte Gruppenprüfung. Den Ablauf haben wir für Sie in Abbildung 8.6 dargestellt. Die Person, die die Inspektion leitet, darf ebenfalls nicht der Autor der zu prüfenden Softwarekomponente sein. Es bedarf einer gründlichen Vorbereitung und Nachbereitung der Inspektion. Die Inspektion basiert auf Regeln, Checklisten, Eingangs- und Ausgangskriterien. Die Initiative für eine Inspektion geht oft vom Management aus. Benannt werden eine Person, die moderiert, und eine, die prüft. Die Prüfer bekommen die Checklisten, mit deren Hilfe die Prüfung der Softwarekomponente stattfindet. Die Diskussion der Ergebnisse erfolgt während der Sitzung. Die Moderatoren müssen auf einen geordneten und zügigen Ablauf achten. Die Sitzung sollte nicht länger als zwei Stunden dauern! In der sogenannten folgenden »dritten Stunde« kann optional über weitere Lösungsansätze und Varianten gesprochen werden. Nach der Sitzung ist die Inspektion noch nicht zu Ende. In der folgenden Nachbereitung wird entschieden, wie mit den gefundenen Schwachstellen und Fehlern umgegangen wird.

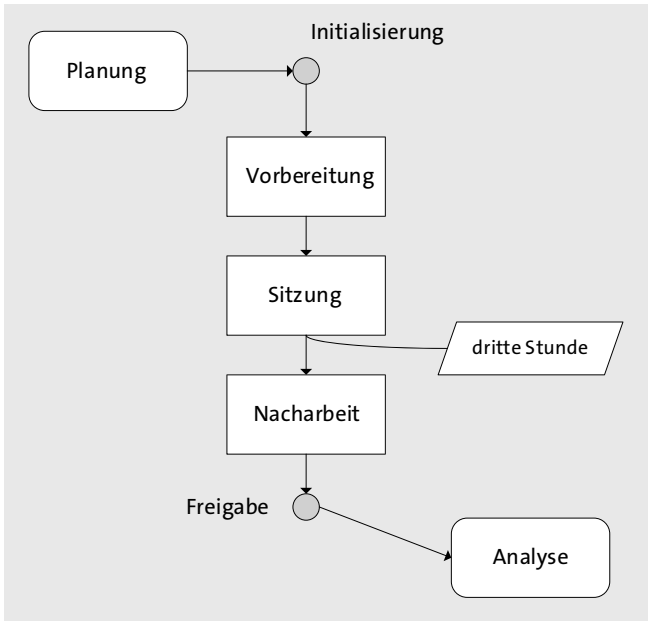

Abbildung 8.6 Phasen einer Inspektion (Quelle: von Delft, 2006)

Vergessen wir nicht: Der Wert solcher Gruppenprüfungen besteht nicht nur darin, dass so die Qualität des Quellcodes begutachtet und idealerweise gesteigert wird, sondern auch darin, dass die Zusammenarbeit im Team gefördert wird. Dabei geht es niemals darum, Autoren beim Auffinden von Fehlern »vorzuführen«. Fehler sind da, damit man sie entdeckt, und zwar möglichst früh – und nicht erst während des späteren praktischen Einsatzes des Produkts. Notwendig ist eine gute und gesunde Fehlerkultur. Wir haben das Thema in diesem Handbuch in Abschnitt 19.5, »Eine gesunde Fehlerkultur«, nochmals aufgegriffen. Eine Review in Form der Retrospektive ist zum Beispiel essenzieller Bestandteil der agilen Methode *Scrum* (siehe Abschnitt 4.5.3 »Scrum«).

8.4.2 Dynamische Tests

Diese Tests werden zur Laufzeit des Programms durchgeführt. Sie setzen also voraus, dass der Quellcode so weit fehlerfrei ist, dass das Programm gestartet werden kann. Da jedoch dynamische Softwaretests bereits stattfinden sollten, wenn das komplette Softwaresystem noch nicht fertig ist, werden auf diese Weise auch einzelne Bestandteile der Software (Komponenten) getestet. Dynamische Tests haben daher oft den Charakter eines Komponententests. Um einen einzelnen Teil einer Software zu testen, muss diese in einen sogenannten *Testrahmen* eingebettet werden (Abbildung 8.7).

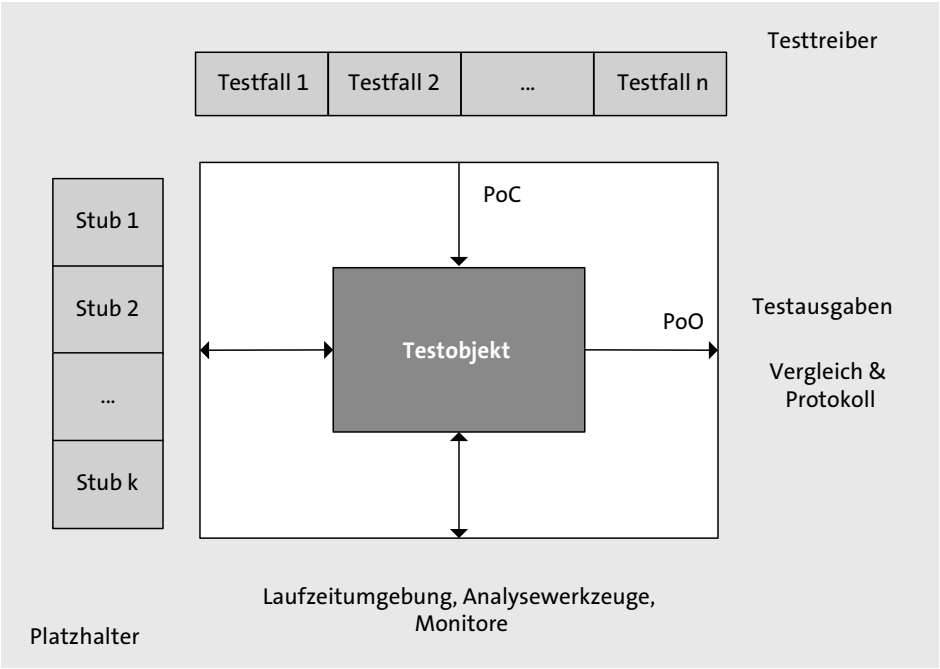

Abbildung 8.7 Testrahmen um ein Testobjekt (Quelle: Spillner und Linz, 2005)

Was nützt uns ein solcher Testrahmen konkret? Der Testrahmen simuliert für das Testobjekt die Programmteile, die noch nicht vorhanden sind bzw. die der Nutzung im Rahmen des separaten Komponententests nicht zur Verfügung stehen. Das Testobjekt wird über definierte Schnittstellen aufgerufen. Man bezeichnet diesen Testrahmen auch als *Testtreiber*. Das *Testobjekt* wird in unterschiedlichen Fallkonstellationen (Testfällen) aufgerufen und durch den Testtreiber mit Daten versorgt. Die Schnittstelle zwischen Testreiber und Testobjekt wird als *Point of Control* (*PoC*) bezeichnet. Die Programmteile, die das Testobjekt aufrufen, bezeichnet man als *Platzhalter* bzw. *Stellvertreter*, *Stub*, *Dummy* oder *Mock*. Sie simulieren das spätere Verhalten des umgebenden Softwaresystems. Die Ausgaben des Testobjekts und der Ist-Soll-Vergleich finden am *Point of Observation* (*PoO*) statt. Der Testrahmen wird noch durch Analyse- und Monitoring-Werkzeuge angereichert, die den Vorgang des Testens erleichtern sollen.

Auf diese Weise können wir einzelne Softwarekomponenten losgelöst von den anderen Programmteilen der Anwendung testen. Die Entwicklung der betreffenden Softwarekomponente (inklusive Test) kann also abgeschlossen werden, bevor das gesamte Softwaresystem fertig ist. Ebenso stellen wir auf diese Weise sicher, dass nur umfassend getestete Softwarebestandteile in das komplette Anwendungssystem übernommen werden.

Bei den dynamischen Tests unterscheidet man

- Black-Box-Tests,
- White-Box-Tests und
- Grey-Box-Tests.

Wir erläutern nun die Unterschiede.

Black-Box-Tests

Bei diesem Test interessiert nur das Ergebnis. Ist es korrekt, kann der Test als erfolgreich bezeichnet werden. Die technischen Abläufe des untersuchten Systems sind nicht von Interesse. Diese Art von Tests sagt nichts über die innere Stabilität und Komplexität der Black-Box aus. Treten Fehler beim Black-Box-Test auf, ist eine weitergehende Analyse notwendig. Fremde Komponenten und Klassen können wir oftmals nur einem solchen Test unterziehen, da der Sourcecode nicht verfügbar ist. Das sollten wir beim Einsatz von fertigen Komponenten und Modulen stets bedenken. Kaufen wir also fertige Komponenten, müssen wir unterscheiden, ob uns auch der Quellcode der Komponenten mit zur Verfügung gestellt wird und ob im Zweifelsfall von uns dort Anpassungen unter Beachtung der Lizenzbestimmungen vorgenommen werden dürfen.

Wie Abbildung 8.8 zeigt, wird das Testobjekt bei einem Black-Box-Test von außen beobachtet.

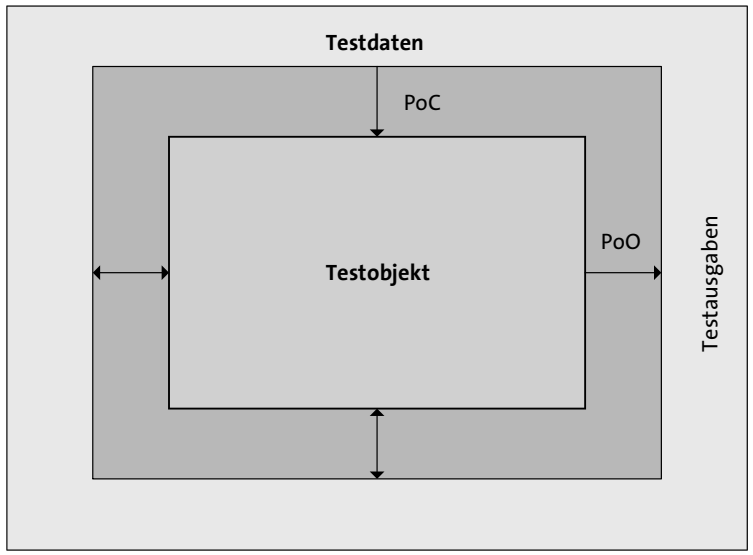

Abbildung 8.8 Testrahmen beim Black-Box-Verfahren (Quelle: Spillner und Linz, 2005)

Bei dieser Art von Test liegen also sowohl der Point of Control (PoC) als auch der Point of Observation (PoO) außerhalb des zu testenden Objekts. Die Testfälle ergeben sich daher aus der Spezifikation, denn sie beantworten die Frage: Was soll das Testobjekt leisten? Bei einem Black-Box-Test muss man also beachten, dass sich der Testumfang auf die vom Tester festgelegte Spezifikation beschränkt. Ist diese Spezifikation nicht (vollständig) korrekt, kann zwar der Test fehlerfrei ablaufen, aber dieser beantwortet nicht die Frage nach der Korrektheit der Softwarekomponente. Ebenso kann das Textobjekt Funktionen enthalten, die nicht durch die aktuelle Spezifikation genutzt und damit geprüft werden. Dieser Umstand ist bei der Wiederwendung von komplexen Komponenten wichtig. Eine mittels Black-Box-Verfahren teilweise getestete Komponente kann bei andersartiger Verwendung zu Fehlern führen. Sie muss erneut getestet werden!

Konkrete Ausprägungen eines *Black-Box-Tests* sind der *Äquivalenzklassentest* und. die *Grenzwertanalyse*. Üblicherweise stellen die Grenzwerte von Daten eine Fehlerquelle dar. Es sind diejenigen Werte, die an den Grenzen des gültigen Wertebereichs der – oft numerischen – Daten liegen. Dazu ein Beispiel:

Der Wertebereich für eine Eingabe der Variablen `Value` beträgt $2 \leqslant$ `Value` < 100. Folgende Testwerte aus den unterschiedlichen Äquivalenzklassen sollten auf jeden Fall in einem Test überprüft werden:

- Ein Wert, der zwischen 2 und 100 liegt, also zum Beispiel `Value = 20`. Der Wert sollte zu einem positiven Testergebnis führen.

- Der Wert `Value = 2` entspricht der unteren Grenze. Auch hier sollte ein positives Testergebnis ermittelt werden.
- Der Wert `Value = 100` entspricht der oberen Grenze und sollte abgewiesen werden.
- Ein Wert von `Value > 100` sollte abgewiesen werden.
- Ein Wert von `Value < 2` sollte auch ein negatives Ergebnis liefern.
- Gegebenenfalls noch ein negativer Wert. Dies ist sinnvoll, um die Verarbeitung negativer Werte zu überprüfen.

Mit der Auswahl der Werte aus verschiedenen Äquivalenzklassen erreicht man, dass unterschiedliche Fallkonstellationen getestet werden. Des Weiteren sollten Werte rund um die Grenzwerte, zum Beispiel 1,99 oder 99,99 oder 100,01, getestet werden. Die Frage ist also: Werden Kommazahlen richtig verarbeitet?

Wie kann es zu Fehlern im Quellcode kommen? Die Ursachen liegen in verwechselten Vergleichsoperatoren (< statt <=) und in Rundungs- oder Konvertierungsfehlern zwischen den Datentypen. Gerade bei der Verarbeitung von Brüchen ist zu beachten, dass der Rechner intern stets mit Näherungswerten arbeitet. Dies kann dazu führen, dass bestimmte Programmteile nicht ausgeführt werden, da der zugehörige Test, zum Beispiel `if Value <= Wert`, niemals positiv bzw. negativ ausfällt. Abhilfe erreicht man, indem entsprechende Fehlerschranken integriert werden.

White-Box-Tests

Es wird versucht, aufgrund der inneren Struktur des Testobjekts alle möglichen Ausführungspfade zu überprüfen und so mögliches Fehlverhalten festzustellen. Im Fokus der Untersuchung steht damit die innere Struktur einer Softwarekomponente, zum Beispiel eine Klasse. Der genaue Testablauf wird anhand der Programmstruktur und des Quellcodes entworfen. Mit den erzeugten Testfällen soll festgestellt werden, ob die erstellten Schleifen, die Verzweigungen und die sonstigen Programmstrukturen zuverlässig funktionieren. Man spricht auch von einer *Überprüfung der Überdeckungsgrade*. Der Testrahmen ist in Abbildung 8.9 zu sehen.

Hier liegen sowohl der Point of Control (PoC) als auch der Point of Observation (PoO) innerhalb des Testobjekts. Voraussetzung für das Durchführen eines White-Box-Tests ist also, dass der Quellcode bekannt ist. Das Ziel ist es, alle Quellcodeteile bei den Tests mindestens einmal auszuführen. Die Testfälle ergeben sich hier aus der Logik des Quellcodes. Die Umsetzung der Spezifikation, d. h. der Anforderungen in Code, wird dabei als gegeben und abgeschlossen vorausgesetzt und steht damit nicht im Fokus des Tests.

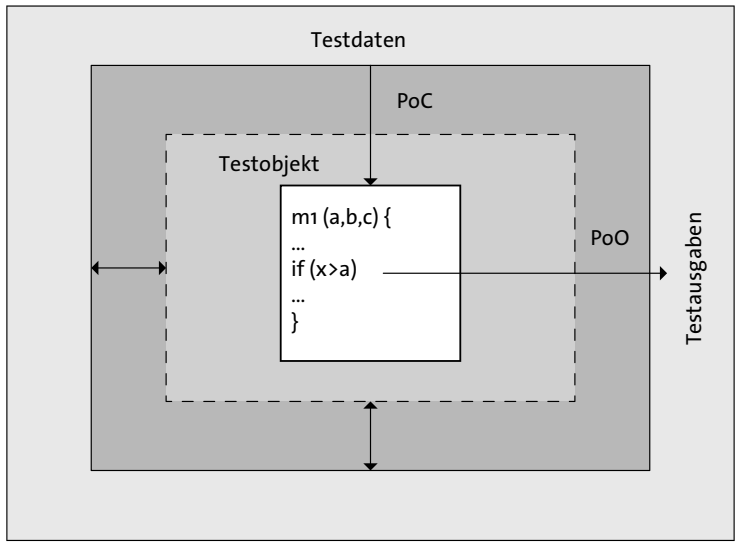

Abbildung 8.9 Testrahmen beim White-Box-Verfahren (Quelle: Spillner und Linz, 2005)

Grey-Box-Tests

Bei dieser Art von Tests wird versucht, White-Box- und Black-Box-Tests zu kombinieren (Abbildung 8.10).

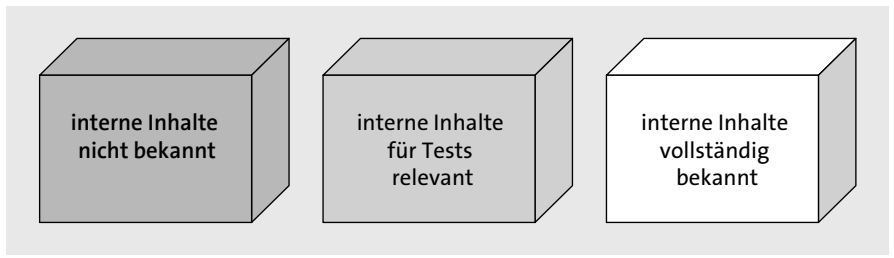

Abbildung 8.10 Verhältnis von Black-, Grey- und White-Box-Test (in Anlehnung an http://www.softwaretestinggenius.com/comparison-among-black-box-testing-and-gray-box-testing-and-white-box-testing)

Voraussetzung ist, dass der Tester die innere Struktur der Softwarekomponente kennt. Die Gestaltung des Tests von außen, also im Sinne eines Black-Box-Tests, findet dann unter Kenntnis dieser inneren Struktur statt. Dadurch können beispielsweise der Testumfang und die Aussagekraft des Tests erhöht werden. Ein Grey-Box-Test kann alle möglichen Codepfade im Inneren der Komponente überprüfen, indem die Testdaten entsprechend ausgewählt werden. Mit anderen Worten: Bei einem Grey-Box-Test überprüfen wir die Funktionalität der Komponente und berücksichtigen dabei ihren inneren Aufbau. Es handelt es sich daher weniger um eine eigene Testmethode als vielmehr um eine Einstellung des Testers.