1 Einführung in die Testautomatisierung und ihre Ziele

Die Softwareentwicklung als Ganzes erlebt eine Entwicklung hin zu einer industriellen Disziplin. Die zunehmende Digitalisierung der Geschäftsprozesse sowie die vermehrte Verbreitung von Standardprodukten und -services sind wesentliche Treiber für den Einsatz von immer effizienteren und effektiveren Methoden im Softwaretest, also auch in der Testautomatisierung. Die rasante Expansion mobiler Applikationen und die sich stetig ändernde Vielfalt der Endgeräte prägen diese Entwicklung ebenfalls nachhaltig.

1.1 Einleitung

Ein wesentliches Merkmal der fortschreitenden Industrialisierung seit dem Ende des 18. Jahrhunderts ist die Mechanisierung energie- oder zeitaufwendiger manueller Tätigkeiten in fast allen Produktionsprozessen. Was vor mehr als 200 Jahren mit mechanischen Webstühlen und Dampfmaschinen in den Textilfabriken Englands begann, ist heute das höchste Ziel und gelebte Praxis in allen produzierenden Unternehmen: die kontinuierliche Steigerung und Optimierung der Produktivität. Ziel ist es stets, mit möglichst geringem Mitteleinsatz das gewünschte Ergebnis in Quantität, Qualität und Zeit zu erreichen. Der Einsatz von Ressourcen bezieht sich sowohl auf den Einsatz menschlicher Arbeitskraft als auch auf den Einsatz von Maschinen, Arbeitsmitteln und anderen (Energie-)Ressourcen.

Softwareentwicklung und Softwaretest auf dem Weg zur industriellen Produktionsreife

Im Bestreben, immer besser zu werden und im globalen Konkurrenzdruck zu bestehen, gibt es keinen Industriebetrieb, der sich nicht laufend mit Optimierungen im Produktionsprozess beschäftigen muss. Vorbild und bestes Beispiel dafür ist die Automobilindustrie, die zu den Themen Prozesssteuerung, Produktionsgestaltung und -messung sowie Qualitätsmanagement immer wieder neue Ideen und Ansätze – auch für andere Industriezweige – hervorgebracht hat und auch weiter hervor-

bringt. Ein Blick in die Produktions- und Fertigungshallen eines Automobilherstellers beeindruckt durch die Präzision des Zusammenspiels zwischen Mensch und Maschine sowie den reibungslosen, hochautomatisierten Fabrikationsablauf. Ein ähnliches Bild zeigt sich mittlerweile in vielen anderen Produktionsprozessen.

Eine nicht unbedingt rühmliche Ausnahme stellt jedoch die Industrie der Softwareentwicklung dar. Trotz vieler Verbesserungen und Bemühungen der letzten Jahre und Jahrzehnte ist diese von der Professionalität der Fertigungsprozesse anderer Branchen immer noch weit entfernt. Dies ist insofern verwunderlich, wenn nicht sogar bedenklich, als Software jene Technologie ist, die in den letzten Jahrzehnten wohl die größte Auswirkung auf gesellschaftliche, wirtschaftliche und technische Veränderungen hatte. Vielleicht liegt es daran, dass die Softwareindustrie noch eine junge Disziplin ist und somit auch nicht die Reife anderer erreichen konnte. Vielleicht liegt es auch am immateriellen Charakter von Softwaresystemen mit all der technologischen Vielfalt, die es generell schwierig macht, Standards zu definieren und konsequent umzusetzen. Oder vielleicht liegt es daran, dass viele die Softwareentwicklung immer noch mehr im Zusammenhang mit künstlerischer Kreativität sehen als mit einer Ingenieurdisziplin.

Auch in den internationalen Standards hat sich die Softwareentwicklung erst als industrieller Zweig etablieren müssen. So findet sich z. B. in der Revision 4 der International Standard Industrial Classification of All Economic Activities (ISIC), veröffentlicht im August 2008, die neue Sektion J »Information and Communication«, während in der Vorgängerversion des Standards die Softwareentwicklungsleistungen noch auf unterster Ebene einer Sektion »Real estate, renting and business activities« versteckt waren [ISIC 08; NACE 08].

Softwareentwicklung als industrielle Einzelfertigung

Das Argument der jungen Disziplin wird jedoch jedes Jahr schwächer. Dennoch wird Softwareentwicklung immer noch gerne als eher künstlerische als ingenieurmäßige Tätigkeit gesehen und wäre demnach anders zu bewerten als die identische Reproduktion zigtausender Türbeschläge. Aber auch wenn Softwareentwicklung nicht den Prozessen einer Massenfertigung unterliegt, so ist diese heutzutage dennoch als industrielle Einzelfertigung zu definieren.

Was bedeutet aber »industriell« in diesem Zusammenhang? Ein industrieller Prozess ist durch mehrere Merkmale gekennzeichnet, insbesondere durch die breite Anwendung von Standards und Normen, den intensiven Einsatz von Mechanisierung und den Umstand, dass es meist um die Bewältigung von großen Mengen und Massen geht. Entlang dieser Merkmale wird auch die Wandlung der Softwareentwicklung von der Kunst hin zu einer professionellen Disziplin deutlich.

1.1.1 Standards und Normen

Seit den Anfängen der Softwareentwicklung gab es eine ganze Reihe von Ansätzen auf der Suche nach dem idealen Entwicklungsprozess. Viele dieser Ansätze waren für ihre Zeit und Rahmenbedingungen zweckmäßig und »State of the Art«. Die rasante Entwicklung der technischen Innovationen, die exponentielle Zunahme fachlicher und anwendungsbezogener Komplexitäten und die ständig wachsenden wirtschaftlichen Herausforderungen bedingen eine laufende Anpassung der bei der Softwareentwicklung eingesetzten Verfahren, Sprachen und Vorgehensmodelle: Wasserfall, V-Modell, iterative und agile Softwareentwicklung; ISO 9001:2008, ISO 15504 (SPICE), CMMI, ITIL; unstrukturierte, strukturierte, objektorientierte Programmierung, ISO/IEC/IEEE 29119 Software Testing – und das Ende der Fahnenstange ist noch lange nicht erreicht. Auch die Disziplin des Softwaretestens im engeren Sinne hat sich vor allem in den letzten Jahren stark verändert. Seit der Gründung des International Software Testing Qualifications Board (ISTQB®) im November 2002 und der damit initiierten, standardisierten Ausbildung zum Certified Tester in den verschiedenen Modulen und Ausbildungsstufen hat sich ein neues Berufs- und Rollenbild des Softwaretesters entwickelt und international etabliert [URL: ISTQB]. Das ISTQB® Ausbildungsprogramm wurde und wird laufend erweitert und aktualisiert und umfasst mit Stand 2020 folgendes Portfolio:

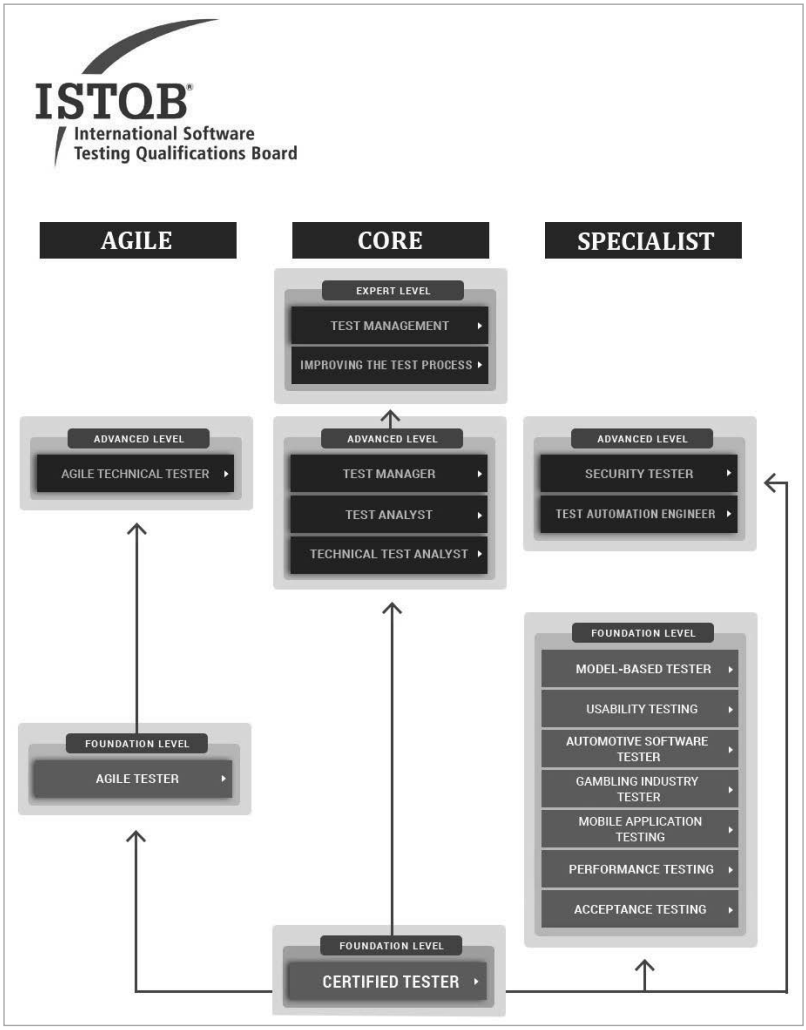

Abb. 1–1
ISTQB®-Produktportfolio (Stand 2020)

Dennoch steckt die Disziplin des Softwaretestens im Vergleich zu anderen Ingenieurdisziplinen und ihrer meist Jahrhunderte oder Jahrtausende alten Tradition und Entwicklung noch in den Kinderschuhen. Dies betrifft nicht nur die inhaltliche Ausgestaltung oder etwa die Durchdringung in Lehre und Praxis.

Einer der Hauptgründe, warum Softwareprojekte trotz vieler Erfahrungen, die sich in Standards manifestieren, immer noch in großem Stil – nachvollziehbar oder nicht – in den Misserfolg schlittern, ist die Tatsache, dass alle bekannten »Best Practices« der Softwareentwicklung weitgehend unverbindlich sind. Wer heute ein Softwareprodukt bestellt, kann sich nicht a priori auf einen überprüfbaren Fertigungsstandard verlassen.

Es ist nicht nur so, dass grundsätzlich jedes Unternehmen selbst entscheidet, ob es bestimmte Produkt- und Entwicklungsstandards anwendet oder nicht, sondern es ist vielerorts auch eine geduldete Praxis, dass diese Unverbindlichkeit auch innerhalb des Unternehmens fortgeführt wird: Jedes Projekt ist ja anders. Das »not invented here«-Syndrom ist und bleibt ein ständiger Begleiter von Softwareentwicklungsprojekten [Katz & Allen 1982].

Noch fehlen in der Testautomatisierung vielfach Normen und Standards.

Auch im Bereich der Testautomatisierung unterliegen technische Konzepte eher selten allgemeinen Normen. Vielmehr bestimmen die Hersteller kommerzieller Werkzeuge oder Open-Source-Communitys den aktuellen Stand der Technik. Diesen beiden Parteien geht es jedoch weniger um die Schaffung eines allgemein gültigen Standards als vielmehr um die Generierung eines Wettbewerbsvorteils am Markt oder die Umsetzung kollektiver Ideen. Denn durch Standards werden Werkzeuge grundsätzlich austauschbar – und welches Unternehmen möchte schon gerne seine Marktposition durch die Schaffung eines Standards schwächen? Eine Ausnahme stellt hier die Testing and Test Control Notation (TTCN-3) des European Telecommunication Standards Institute (ETSI) dar [URL: ETSI]. Die Anwendung dieses Standards ist in der Praxis jedoch im Wesentlichen auf sehr spezifische Einsatzgebiete beschränkt, zum Beispiel in den Telekommunikations- und Automotive-Sektoren.

Für ein Unternehmen, das eine Testautomatisierung einführt, bedeutet dies in der Regel eine enge Bindung an einen Werkzeughersteller. Auch in Zukunft wird es nicht möglich sein, eine umfassende, automatisierte Testsuite einfach von einem Werkzeug auf ein anderes zu übertragen, da sich gemeinhin sowohl die technologischen Konzepte als auch die Automatisierungsansätze stark unterscheiden. Dies gilt auch für die Investitionen in die Ausbildung des Personals, die ebenfalls eine stark werkzeugbezogene Komponente enthält.

Dennoch gibt es einige allgemeine Prinzipien bei der Konzeption, Organisation und Durchführung von automatisierten Softwaretests. Diese Faktoren unterstützen dabei, die Abhängigkeit von bestimmten Werkzeugen zu reduzieren und die Produktivität bei der Automatisierung zu optimieren.

Die Ausbildung zum ISTQB® Certified Tester Advanced Level Testautomatisierungsentwickler und dieses Buch, in das die Autoren auch ihre praktischen Erfahrungen eingebracht haben, bieten eine Einführung in diese grundsätzlichen Aspekte und Prinzipien sowie eine Anleitung bzw. Empfehlungen zur Umsetzung eines Testautomatisierungsprojekts.

1.1.2 Der Einsatz von Maschinen

Ein weiteres wesentliches Merkmal der industriellen Fertigung ist der Einsatz von Maschinen zur Erleichterung und Reduktion von manuellen Tätigkeiten. In der Softwareentwicklung sind diese Maschinen selbst wiederum Software. Dazu zählen etwa Entwicklungsumgebungen, die die Erstellung und Verwaltung des Programmcodes und weiterer Softwarebestandteile vereinfachen bzw. erst ermöglichen. Im Prinzip sind dies jedoch meist nur Bearbeitungs- und Verwaltungssysteme mit gewissen Kontrollmechanismen, wie sie etwa ein Compiler durchführt. Die Programme müssen immer noch »mit Hand und Verstand« geschrieben werden. Eine »Mechanisierung« der Programmierung ist das Ziel der modellbasierten Ansätze, in denen die mühsame Arbeit der Codierung von Generatoren erledigt wird. Ausgangsbasis für die Codegenerierung sind Modelle des zu entwickelnden Softwaresystems, z.B. in der UML-Notation. In einigen Bereichen wird diese Technologie bereits umfassend eingesetzt, etwa zur Generierung von Datenzugriffsroutinen oder dort, wo Spezifikationen in formalen Sprachen vorliegen, wie etwa bei der Entwicklung von eingebetteten Systemen. In der Breite jedoch ist Softwareentwicklung immer noch pures Handwerk.

Die Mechanisierung im Softwaretest

Einsatz von Werkzeugen zur Testfallgenerierung und Testdurchführung

Eine Aufgabe des Softwaretesters ist die Identifikation von Testbedingungen und der Entwurf von Testfällen. Ähnlich den Ansätzen zur modellbasierten Entwicklung zielt das modellbasierte Testen (MBT) auf die automatische Ableitung von Testfällen aus vorhandenen Modellbeschreibungen des Systems unter Test (SUT) ab. Als Ausgangsbasis dienen z.B. Objektmodelle, Use-Case-Beschreibungen und Ablaufgraphen in verschiedenen Notationen. Durch Anwendung eines semantischen Regelwerks werden fachliche Testfälle auf Basis textueller Spezifikationen abgeleitet. Und nicht zuletzt generieren entsprechende Parser aus dem Quellcode selbst abstrakte Testfälle, die dann zu konkreten Testfällen verfeinert werden. Für die Verwaltung dieser Testfälle steht eine Vielzahl geeigneter Testmanagementwerkzeuge zur Verfügung, die in die unterschiedlichen Entwicklungsumgebungen integriert sind. Ähnlich der Generierung von Code aus den Modellen ist auch die Generierung von Testfällen aus Testmodellen noch nicht sehr weit verbreitet. Ein Grund dafür ist der Umstand, dass das Ergebnis, der generierte Testfall, im hohen Grad von der Qualität und der »passenden« Beschreibungstiefe der Modelle abhängt. Diese sind zumeist nicht zufriedenstellend gegeben.

Eine weitere Aufgabe des Softwaretesters ist die Durchführung und Protokollierung der Testfälle. An dieser Stelle ist zu unterscheiden, ob es sich um Tests handelt, die auf der Ebene technischer Schnittstellen, Systemkomponenten, Module oder Methoden durchgeführt werden, oder um fachliche, anwenderbezogene Tests, die eher über die Benutzerschnittstelle erfolgen. Für erstere werden bereits vorwiegend technische Hilfsmittel eingesetzt: Testrahmen, Testtreiber, Unit-Test-Frameworks, Hilfsprogramme etc. Diese Tests werden auch fast immer von »Technikern« durchgeführt, die sich die »mechanischen Hilfsmittel« selbst bereitstellen können. Der fachliche Test wurde und wird zum überwiegenden Teil von Mitarbeitern aus den Fachbereichen oder dedizierten Testanalysten manuell ausgeführt. Für diesen Bereich stehen ebenfalls Werkzeuge zur Unterstützung und Erleichterung der manuellen Testdurchführungen zur Verfügung. Deren Einsatz ist jedoch mit entsprechenden Kosten und Lernaufwänden verbunden. Dies ist mit ein Grund dafür, dass der Einsatz von Testautomatisierungswerkzeugen in der Vergangenheit keine allgemeine Verbreitung gefunden hat. Die Weiterentwicklung dieser Werkzeuge hat in den letzten Jahren jedoch zu einer deutlichen Verbesserung der Kosten-Nutzen-Relation geführt. Die vereinfachte Erstellung eines automatisierten Testfalls sowie seine einfachere Wartbarkeit durch die zunehmende Trennung von fachlicher Logik und technischer Implementierung führten dazu, dass sich Automatisierung nicht erst dann rechnet, wenn riesige Mengen von Testfällen durchzuführen oder der x-te Regressionstest zu wiederholen sind, sondern bereits bei der erstmaligen automatisierten Durchführung von manuell aufwendigen Tests.

1.1.3 Mengen und Massen

Während in der Programmierung eine beschränkte Anzahl von Programmen oder Objekten und Methoden einmal entwickelt wird und dann bestenfalls noch adaptiert bzw. korrigiert werden muss, ist im Test theoretisch eine fast grenzenlose Anzahl an Testfällen möglich. In der Praxis gehen die Testfallanzahlen meist in die Hunderte oder Tausende. Eine einmal entwickelte Eingabemaske und ein einmal entwickelter Verarbeitungsalgorithmus muss im Test unzählige Male getestet werden, z.B. durch verschiedene Eingabe- und Dialogvariationen oder etwa durch die Erfassung von hundert Verträgen mit unterschiedlichen Tarifen im datengetriebenen Test. Diese Tests sind aber nicht nur ein einziges Mal zu erstellen und durchzuführen. Mit jeder Änderung am System müssen Regressionstests durchgeführt und angepasst werden, um die Funktionsfähigkeit des Systems nachzuweisen. Um

eventuell entstandene Nebeneffekte von Änderungen zu erkennen, ist jedes Mal eine möglichst große Testüberdeckung anzustreben, die aber erfahrungsgemäß aus Kosten- und Zeitgründen meist nicht erreicht werden kann.

Der notwendige Testumfang kann nur mit dem Einsatz von Automaten bewältigt werden.

Diese Ausgangslage, die notwendige Bewältigung von großen Mengen und Massen, schreit förmlich nach dem Einsatz der industriellen Mechanisierung, nach dem Einsatz von Testautomatisierungslösungen. Und wenn die Lage nicht schreit, so tun dies die Tester, die im Gegensatz zu einer Maschine bei der zehnten Durchführung desselben Testfalls menschliche Reaktionen wie Frustration, mangelnde Konzentration oder Ungeduld zeigen. Individuelle Priorisierungen lassen dann unter Umständen gerade die falschen, weil erfolgskritischsten Testfälle unter den Tisch fallen.

Betrachtet man die obigen Punkte, ist es eigentlich verwunderlich, dass Testautomatisierung nicht schon längst durchgängig im Einsatz ist. Fehlende Standardisierungen, bisherige unattraktive Kosten-Nutzen-Rechnungen und der Entwicklungsstand der Werkzeuge mögen Gründe dafür gewesen sein. Heutzutage führt jedoch an der Testautomatisierung kein Weg mehr vorbei. Der Anstieg der Komplexität der Softwaresysteme und der dadurch erhöhte Testbedarf, der steigende Druck auf Zeit und Kosten sowie die wachsende Verbreitung agiler Entwicklungsansätze und mobiler Applikationen zwingen Unternehmen regelrecht, bei der Softwareentwicklung auf eine nachhaltige Testautomatisierung zu setzen.

1.2 Was ist unter Testautomatisierung zu verstehen?

In der Definition des ISTQB® ist unter Testautomatisierung der »Einsatz von Softwarewerkzeugen zur Durchführung oder Unterstützung von Testaktivitäten, z. B. Testmanagement, Testentwurf, Testausführung und Soll/Ist-Vergleich« zu verstehen. Man könnte auch sagen, »Testautomatisierung ist die Durchführung von ansonsten manuellen Testtätigkeiten durch Automaten bzw. Maschinen«. Das Spektrum umfasst demnach alle Tätigkeiten zur Überprüfung der Softwarequalität im Entwicklungsprozess, in den unterschiedlichen Entwicklungsphasen und Teststufen sowie die entsprechenden Aktivitäten von Entwicklern, Testern, Analytikern oder auch der in die Entwicklung eingebundenen Anwender.

Testautomatisierung bedeutet demnach nicht nur die Ausführung einer Testsuite, sondern umfasst den gesamten Prozess der Bereitstellung und Erstellung aller Testmittel, d. h. der Arbeitsergebnisse, die für die Planung, den Entwurf, die Ausführung, die Auswertung und die Berichterstattung über automatisierte Tests erforderlich sind.

Relevante Testmittel sind unter anderem:

- **Software**
  Für die Verwaltung, den Entwurf, die Implementierung, die Durchführung und die Auswertung von automatisierten Testsuiten ist eine Reihe von Werkzeugen (Automatisierungswerkzeuge, Testrahmen, Virtualisierungslösungen etc.) erforderlich. Die Auswahl und Bereitstellung dieser Werkzeuge ist eine komplexe Aufgabe, die von der Technologie und dem Umfang des SUT und der gewählten Testautomatisierungsstrategie abhängt.
- **Dokumentation**
  Dazu gehört nicht nur die Dokumentation der verwendeten Testwerkzeuge, sondern auch alle verfügbaren fachlichen und technischen Beschreibungen sowie der Architektur und der Schnittstellen des SUT.
- **Testfälle**
  Die fachlichen Testfälle, abstrakt oder konkret, bilden die Grundlage für die Implementierung automatisierter Tests. Deren Auswahl bzw. Priorisierung und funktionale Qualität (z.B. fachliche Relevanz, funktionale Überdeckung, Korrektheit) sowie die Qualität ihrer Beschreibung haben einen wesentlichen Einfluss auf das langfristige Kosten-Nutzen-Verhältnis der Testautomatisierungslösung (TAS – Test Automation Solution) und damit unmittelbar auf ihre Nachhaltigkeit.
- **Testdaten**
  Die Testdaten sind der Treibstoff für die Testdurchführung. Sie werden zur Steuerung der Testszenarien sowie zur Berechnung und Verifizierung der Testergebnisse verwendet. Sie stellen dynamische Eingabedaten, feste oder variable Parameter und (Konfigurations-)Daten dar, auf denen die Verarbeitung basiert. Die Erzeugung, Produktion und Wiederherstellung von Bestands- und Verarbeitungsdaten für und durch die Testautomatisierung erfordert besondere Aufmerksamkeit. Falsche Testdaten, wie fehlerhafte Testskripte, führen zu falschen Testergebnissen und können den Testfortschritt stark behindern. Auf der anderen Seite bieten die Testdaten die Möglichkeit, das Potenzial der Testautomatisierung voll auszuschöpfen. Die Bedeutung, aber auch die Komplexität eines effizienten und gut organisierten Testdatenmanagements spiegelt sich auch in der Ausbildung zum GTB Certified Tester Foundation Level Test Data Specialist [GTB 18] wider.

- **Testumgebungen**
  Der Aufbau von Testumgebungen ist in der Regel eine sehr komplexe Aufgabe und natürlich stark abhängig von der Komplexität des SUT sowie von den technischen und organisatorischen Rahmenbedingungen im Unternehmen. Es ist daher wichtig, den Betrieb, das Testumgebungsmanagement, das Applikationsmanagement etc. rechtzeitig mit allen Beteiligten zu diskutieren. Es ist zu klären, wer und in welcher Form das SUT, die benötigten Drittsysteme, die Datenbanken und die Testautomatisierungslösung selbst in der Testumgebung bereitstellt, alle erforderlichen Zugriffsrechte gewährt und die Ausführung überwacht.

  Die Testautomatisierungslösung sollte nach Möglichkeit vom SUT getrennt sein, um gegenseitige Beeinflussung zu vermeiden. Ausnahmen sind z.B. eingebettete Systeme, bei denen die Testsoftware in das SUT integriert werden muss.

Obwohl sich der Begriff »Testautomatisierung« grundsätzlich auf alle Aktivitäten im Testprozess bezieht, wird in der Praxis gemeinhin die automatisierte Ausführung von Tests mithilfe spezieller Werkzeuge oder Software mit diesem Begriff in Verbindung gebracht.

Dabei werden eine oder mehrere Aufgaben, wie sie auch für die Durchführung dynamischer Tests definiert sind [Spillner & Linz 19], auf Basis der genannten Testmittel ausgeführt:

- Die automatisierten Testfälle auf der Grundlage der vorhandenen Spezifikationen, der fachlichen Testfälle und des SUT implementieren und sie mit Testdaten versorgen.
- Die definierten Vorbedingungen für die automatisierte Durchführung herstellen und steuern.
- Die automatisierten Testsuiten ausführen, steuern und überwachen.
- Die Ergebnisse der Ausführung – d.h. den Vergleich der tatsächlichen mit den erwarteten Ergebnissen – protokollieren, auswerten und entsprechende Berichte bereitstellen.

Aus technischer Sicht kann die Implementierung von automatisierten Tests auf verschiedenen Architekturebenen ansetzen. Unter dem Gesichtspunkt, die manuelle Testausführung zu ersetzen, greift die Automatisierung auf die Elemente der grafischen Benutzungsoberfläche (GUI-Test) oder – je nach Art der Anwendung – auf die Kommandoschnittstelle des SUT (CLI-Test) zu. Geht man eine Ebene tiefer, kann die Automatisierung über die öffentlichen Schnittstellen der Klassen, Module und Bibliotheken des SUT (API-Test) und über entsprechende Dienste (Servicetest) und Protokolle (Protokolltest) implementiert werden. Test-

fälle, die auf dieser tieferen Architekturebene umgesetzt werden, haben den Vorteil, dass sie weniger empfindlich auf (häufige) Änderungen der Benutzungsschnittstellen reagieren. Neben der wesentlich höheren Wartungsfreundlichkeit hat dieser Ansatz in der Regel auch einen deutlichen Performanzvorteil gegenüber einer GUI-basierten Automatisierung. Selbst vor der eigentlichen Verteilung der Software auf eine Laufzeitumgebung können wertvolle Tests durchgeführt werden: Mit Unit-Tests kann für jeden Build die automatisierte Prüfung der einzelnen Softwarekomponenten durchgeführt werden, noch bevor eine vollständige Integration und Paketierung in ein Softwareprodukt durchgeführt wird. Die Pyramide der Testautomatisierung nach Mike Cohn veranschaulicht die angestrebte Verteilung der automatisierten Tests auf der Grundlage ihrer Kosten-Nutzen-Effizienz über die Zeit [URL: Cohn].

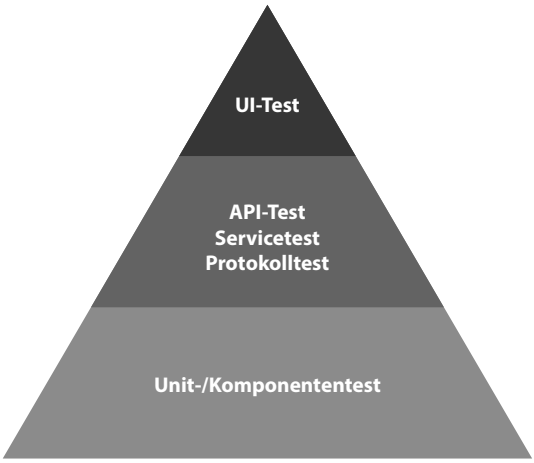

Abb. 1–2
Pyramide der Testautomatisierung

1.3 Ziele der Testautomatisierung

Die Einführung einer Testautomatisierung ist in der Regel mit einer Reihe von Zielen und Erwartungen verbunden – denn bei allen Vorteilen ist und bleibt Automatisierung kein Selbstzweck. Zu den Zielen gehören zunächst einmal die Steigerung der Testeffizienz und damit die Senkung der Gesamtkosten für den Test. Weitere wichtige Faktoren sind die Verkürzung der Testausführungszeit/Testzyklen und die sich daraus ergebende Möglichkeit, die Häufigkeit der Testausführungen zu erhöhen. Dies ist besonders für die Ansätze DevOps und DevTestOps wichtig. Continuous Integration, Continuous Deployment und Continuous Testing können nur mit einer gut funktionierenden Testautomatisierungslösung effektiv realisiert werden.

Neben der Kostenreduktion und Zeitverkürzung ist die Steigerung bzw. Aufrechterhaltung der Qualität ein wichtiges Ziel der Testautomatisierung. Die Qualität kann durch eine Erhöhung der Funktionsüberdeckung und durch die Implementierung von Tests erreicht werden, die nicht oder nur mit großem Ressourcenaufwand manuell durchgeführt werden können. Beispiele hierfür sind der Test einer sehr großen Anzahl relevanter Datenkonstellationen oder -variationen, das Testen auf Fehlertoleranz, d.h. die Testausführung auf API-/Service-Ebene mit fehlerhaften Eingabedaten, um die Robustheit des SUT zu bewerten, oder auch der Performanztest in seinen verschiedenen Ausprägungen. Aber auch die einheitliche und wiederholte Ausführung ganzer Testsuiten gegen verschiedene Versionen des SUT (Regressionstest) in verschiedenen Umgebungen (unterschiedliche Browser und Versionen, Vielzahl mobiler Endgeräte etc.) ist nur unter Einsatz von Testautomatisierung wirtschaftlich machbar.

Vorteile der Testautomatisierung

Einer der größten Vorteile und Nutzen der Testautomatisierung ergibt sich durch den Aufbau einer automatisierten Regressionstestsuite, die es ermöglicht, mehr und mehr Testfälle je Softwarerelease durchzuführen. Ein manueller Regressionstest stößt sehr rasch an die Grenzen der Machbarkeit und Wirtschaftlichkeit. Zudem verliert er durch die nachlassende Konzentration und Motivation der Tester von Mal zu Mal an Effektivität und bindet wertvolle manuelle Ressourcen. Demgegenüber laufen automatisierte Tests schneller ab, sind weniger anfällig für Bedienerfehler und auch komplexere Testszenarien können – einmal konzentriert erstellt – wiederholt durchgeführt werden. Bei der manuellen Testdurchführung muss oft jedes Mal sehr viel Zeit investiert werden, um komplexere Testsequenzen wieder zu verstehen und in gleicher Qualität auszuführen.

Bestimmte Tests sind manuell auch kaum bzw. gar nicht durchführbar. So können automatisiert relativ einfach verteilte und parallele Tests implementiert und durchgeführt werden, zum Beispiel für die Durchführung von Last-, Performanz- und Stresstests. Auch Echtzeittests etwa in der Steuerungstechnik benötigen zwingend entsprechende Werkzeuge.

Da die automatisierten Testfälle und Testszenarien auf Basis eines einheitlichen Frameworks erstellt werden, einheitlich formal beschrieben sind, also im Gegensatz zum manuellen Testfall keinen Interpretationsspielraum zulassen, und quasi immer »stimmen« müssen, erhöhen sich zum einen die Konsistenz und Wiederholbarkeit der Tests und zum anderen die Ausfallsicherheit des SUT.

Auch aus Projektsicht ergeben sich wesentliche Vorteile durch den Einsatz einer Testautomatisierung. Die unmittelbare Rückmeldung bezüglich der Qualität des SUT beschleunigt den Projektverlauf deutlich. Vorhandene Probleme werden innerhalb von Stunden und nicht nach mehreren Tagen oder Wochen aufgezeigt und können behoben werden, bevor die Aufwände für die Korrekturen noch höher werden.

Die Testautomatisierung ermöglicht auch die effizientere und effektive Nutzung von Testressourcen. Damit sind nicht nur technische Infrastrukturen gemeint. Ein großer Vorteil ist das Freiwerden von Testern, auch aus den Fachbereichen, durch die Automatisierung der Regressionstests. Dadurch können sich diese Tester verstärkt der Fehlerfindung z. B. durch exploratives Testen oder dem gezielten Einsatz diverser Testverfahren im dynamischen, manuellen Test widmen.

Nachteile der Testautomatisierung

Natürlich bringt die Testautomatisierung nicht nur Vorteile, sondern auch den einen oder anderen Nachteil mit sich bzw. Aspekte, die man im Vorfeld kennen sollte, um hinterher nicht negativ überrascht zu werden.

Die Automatisierung von Prozessen ist immer mit zusätzlichen Kosten verbunden, die Testautomatisierung bildet hier keine Ausnahme. Zu den Anfangsinvestitionen, die für den Aufbau und die Inbetriebnahme einer Testautomatisierungslösung erforderlich sind, gehören Werkzeuge (bspw. für die Testdurchführung), die angeschafft oder entwickelt werden müssen, die Arbeitsplatzausstattung für die Testautomatisierungsentwickler (TAE), die in der Regel mehrere Rechner oder Bildschirme (Entwicklungsrechner, Ausführungsrechner) umfasst, die Aufrüstung der Testumgebungen, die Etablierung neuer Prozesse und Arbeitsschritte, die für die Entwicklung von Testskripten notwendig werden, zusätzliche Konfigurationsmanagement- und Versionierungssysteme u. v. m.

Neben der Investition in zusätzliche Technologien oder Prozesse müssen auch Zeit und Geld in den Ausbau der Kompetenzen des Testteams investiert werden. Dazu gehören die Ausbildung zum Testautomatisierungsentwickler und die Weiterbildung im Bereich der Softwareentwicklung sowie die Schulung für den Einsatz der Testautomatisierungslösung und der eingesetzten Werkzeuge.

Häufig unterschätzt wird auch der Aufwand für die Wartung der Testautomatisierungslösung und der automatisierten Testmittel, allen voran natürlich der Testskripte. Letztlich erzeugt die Testautomatisierung selbst Software, die gewartet werden muss. Eine ungeeignete Architektur, Nichteinhaltung von Konventionen, unzureichende Dokumentation und fehlendes Konfigurationsmanagement wirken sich dramatisch aus,

sobald die automatisierte Testsuite ein bestimmtes Niveau erreicht hat. Änderungen und Erweiterungen finden immer statt. Das SUT ändert sich an der Benutzerschnittstelle, in den Prozessen, in den technischen Aspekten, den Geschäftsregeln usw. Und diese Änderungen wirken sich direkt und unmittelbar auf die Testautomatisierungslösung oder die automatisierten Testmittel aus.

Es ist nicht ungewöhnlich, dass der Testautomatisierungsentwickler von diesen Änderungen erst »in der Produktion« erfährt, nämlich dann, wenn bei der Testausführung eine Abweichung auftritt. Diese Abweichung wird gemeldet und dann vom Entwickler als Fehler der TAS (ein sogenanntes falsch positives Ergebnis) zurückgewiesen. Aber dies ist nicht das einzige Szenario, in dem die TAS zu Fehlern führt, denn, wie gesagt, die TAS ist auch nur Software – und Software ist in den allermeisten Fällen fehlerhaft.

Aus diesem Grund fokussieren sich die Testautomatisierungsentwickler oft zu sehr auf die technischen Aspekte der TAS und lassen sich von den eigentlichen, qualitativen Testzielen, die für die erforderliche Überdeckung des SUT notwendig sind, ablenken.

Ist eine TAS erst einmal etabliert und leistet gute Dienste, unterliegen Tester allzu gerne der Verlockung, alles automatisieren zu wollen, z. B. umfangreiche Ende-zu-Ende-Tests, ineinander verwobene Dialogsequenzen oder eine komplizierte Verarbeitung. Das klingt nach einer großartigen Sache, aber man muss sich des Aufwands bewusst sein, der mit der Implementierung und Wartung von automatisierten Tests verbunden ist. Allein die Erstellung und Pflege von konsistenten Testdaten über mehrere Systeme hinweg für umfangreiche Ende-zu-Ende-Tests ist in vielen Situationen eine große Herausforderung.

Beschränkungen und Grenzen der Testautomatisierung

Auch die Testautomatisierung hat ihre Grenzen. Technisch gesehen ist vieles möglich, aber manchmal stehen die Kosten für die Automatisierung bestimmter manueller Tests in keinem Verhältnis zum Nutzen.

Ein Automat kann nur tatsächliche, maschineninterpretierbare Ergebnisse prüfen und benötigt dafür ein Testorakel, das ebenfalls automatisiert sein muss. Die Stärke der Testautomatisierung liegt sicherlich in der Verifikation, dem exakten Vergleich von erwartetem mit dem tatsächlichen Verhalten des SUT. Die Schwäche liegt in der Validierung des Systems, der Bewertung der Eignung für die beabsichtigte Nutzung. Fehler in den Anforderungen oder deren fehlerhafte Interpretation werden von der Testautomatisierungslösung nicht erkannt. Ein Test »zwischen den Zeilen« und Testkreativität ist der Testautomatisierungslösung nicht bekannt. Daher ist die Testautomatisierung kein

vollwertiger Ersatz für manuelle strukturierte, dynamische Tests oder für explorative Tests. Das SUT sollte bereits ein gewisses Maß an Fehlerfreiheit und Stabilität an den Benutzer- und Systemschnittstellen erreicht haben, damit die Testabläufe sinnvoll automatisiert werden können und nicht ständigen Änderungen unterliegen.

1.4 Erfolgsfaktoren für die Testautomatisierung

Um die gesteckten Ziele zu erreichen, die Erwartungen langfristig zu erfüllen und die Hindernisse so gering wie möglich zu halten, sind folgende Erfolgsfaktoren für laufende Testautomatisierungsprojekte von besonderer Bedeutung. Und je mehr diese erfüllt sind, umso höher ist die Wahrscheinlichkeit, dass das Testautomatisierungsprojekt erfolgreich sein wird. In der Praxis wird es selten der Fall sein, dass alle Kriterien erfüllt sind, und dies ist auch nicht zwingend erforderlich. Bereits vor Beginn des Projekts müssen die Rahmenbedingungen und Erfolgsaussichten geprüft und während des Projekts laufend analysiert werden. Jeder Ansatz hat im jeweiligen Projektkontext auch Risiken, und man muss sich bewusst sein, welche Erfolgsfaktoren erfüllt sind und welche nicht. Die Testautomatisierungsstrategie und -architektur sind demnach auch stets an veränderte Rahmenbedingungen anzupassen und zu ergänzen.

Auf Erfolgsfaktoren für die Pilotierung von Testautomatisierungsprojekten wird in weiterer Folge nicht eingegangen.

1.4.1 Testautomatisierungsstrategie

Die Testautomatisierungsstrategie ist ein Plan auf hoher Ebene zur Erreichung der langfristigen Ziele der Testautomatisierung unter gegebenen Randbedingungen. Aussagen über die Testautomatisierungsstrategie können in der Testrichtlinie des Unternehmens oder auch in der organisatorischen Teststrategie enthalten sein. Letztere definiert die generischen Anforderungen an das Testen in einem oder mehreren Projekten innerhalb einer Organisation, einschließlich Details darüber, wie das Testen durchgeführt werden soll, und ist an der Testrichtlinie ausgerichtet.

Für jedes Testautomatisierungsprojekt ist eine pragmatische und konsistente Testautomatisierungsstrategie erforderlich, die auf die Wartbarkeit der Testautomatisierungslösung und die Konsistenz des SUT abgestimmt ist.

Da das SUT selbst aus verschiedenen alten und neuen funktionalen und technischen Bereichen bestehen kann und auch Anwendungen und Komponenten verschiedener Plattformen umfasst, ist es wahrscheinlich, dass neben einer Basisstrategie auch entsprechende spezifische Strate-

gien definiert werden müssen. Dabei ist zu berücksichtigen, welche Kosten, Nutzen und Risiken die Anwendung der Strategie auf die verschiedenen Bereiche des SUT mit sich bringt.

Eine weitere wesentliche Anforderung an die Testautomatisierungsstrategie ist die Sicherstellung der Vergleichbarkeit der Testergebnisse von automatisierten Testfällen, die über verschiedene Schnittstellen (z.B. API und GUI) des SUT ausgeführt werden.

Im Laufe des Projekts werden kontinuierlich Erfahrungen gesammelt, das SUT wird sich verändern und die Ziele können angepasst werden. Entsprechend muss die Teststrategie laufend adaptiert und verbessert werden. Deshalb müssen in der Strategie auch die Verbesserungsprozesse und die Strukturen definiert werden.

Exkurs: Testautomatisierungs-Manifest

Es können auch Grundprinzipien für die Testautomatisierung im Projekt oder für das Unternehmen formuliert werden, ein Mindset, das als Leitfaden in verschiedenen Fragestellungen dienen kann. Ein Beispiel dafür ist in der folgenden Abbildung aus dem Projektumfeld der Autoren dargestellt:

Abb. 1–3
Testautomatisierungs-Manifest

Testautomatisierungs-Manifest

Transparenz vor Komfort	Zusammenarbeit vor Unabhängigkeit	Qualität vor Quantität	Flexibilität vor Kontinuität
Die Testautomatisierung muss gut sichtbar sein, um Mehrwert zu generieren. Wir stehen für Transparenz, auch wenn dies bedeutet, dass wir Fehler in unserer eigenen Arbeit aufdecken müssen.	Es ist besser, mit anderen Beteiligten und Organisationen zusammenzuarbeiten, und sich zu vernetzen, als Probleme im Alleingang zu lösen.	Verlässliche Ergebnisse, die die weitere Arbeit voranbringen, sind wichtiger als eine hohe Anzahl von automatisierten Testfällen.	Gegenüber festen Strukturen bevorzugen wir eine flexible Vorgehensweise, die künftigen Herausforderungen standhält.

Transparenz vor Komfort

Testautomatisierung hat den Charakter der Risikomessung und -vermeidung, ähnlich dem Sicherheitsnetz eines Artisten auf einem Hochseil. Das heißt, wenn alle Faktoren richtig zusammenspielen, ist etwa der »Output« im Regressionstest, d.h. die erkannten Fehler, gering. Dies bedeutet jedoch nicht, dass die Testautomatisierung keinen Mehrwert bietet. Es ist daher wichtig, die Testautomatisierung und ihre Ergebnisse und Funktion klar und sichtbar in der Organisation zu platzieren. Dies bedeutet aber auch, dass Probleme der Testautomatisierung klar und sofort sichtbar sind. Dies ist nach Meinung der Autoren kein Nachteil, sondern eine Stärke.

Zusammenarbeit vor Unabhängigkeit

Eine typische Situation für die Testautomatisierung besteht darin, dass ein Werkzeug gekauft und an einen Tester übergeben wird, der dann für die Einführung und Nutzung dieses Werkzeugs verantwortlich ist. Häufig tritt der Effekt auf, dass diese Person dann in einen experimentellen Modus gerät und unter Druck versucht, automatisierte Testfälle zu implementieren. Ein Verhaltensmuster in diesem Zusammenhang ist »ich und das Werkzeug gegen das Produkt« – d.h. eine Tendenz, Probleme und Herausforderungen allein lösen zu wollen oder sie zu umgehen. Wir empfehlen, sich stattdessen aktiv mit anderen Rollen auszutauschen. Wenn z.B. die Anzeige einer Tabelle schwer zugänglich ist, wenden Sie sich an die Entwickler, fragen Sie die Community oder rufen Sie einfach den Herstellersupport an.

Qualität vor Quantität

Eine typische Metrik für den Wert und Fortschritt von Testautomatisierung ist der Automatisierungsgrad einer Testsuite in Prozent oder auch die absolute Anzahl der automatisierten Testfälle. Dabei wird allerdings nicht der generierte Mehrwert betrachtet, der direkt von der Wartbarkeit und Robustheit der automatisierten Tests abhängt. Ein Leitspruch in diesem Kontext ist: »Zehn aussagekräftige, stabile, automatisierte Tests sind mehr wert als tausend instabile und nicht nachvollziehbare Testfälle.« Eine kleine Regressionstestsuite ist also oft nützlicher als ein riesiges, nur schwer wartbares Testportfolio.

Flexibilität vor Kontinuität

Die Testautomatisierung ist ein »Zwillingsprodukt« zu den Systemen, die sie testet – und oft ein Werkzeug zur Sicherstellung von Geschäftsprozessen. Sie liefert den höchsten Mehrwert, wenn sie über einen langen Zeitraum mit geringem Wartungsaufwand betrieben werden kann. In dieser Zeit können sich Technologien, Werkzeuge, Personal und auch die Geschäftsprozesse erheblich ändern. Um effektiv zu bleiben, erfordert die Testautomatisierung ein hohes Maß an Flexibilität gegenüber Veränderungen. Dies ist sowohl ein strategisches und prozessbezogenes als auch ein technologisches/architektonisches Problem, das auch durch die in späteren Kapiteln ausführlich beschriebene generische Testautomatisierungsarchitektur adressiert wird.

Die Testautomatisierungsstrategie muss auch auf den Typ des zugrunde liegenden Projekts abgestimmt sein. Auch die verschiedenen Teststufen und Testarten, die automatisiert unterstützt werden sollen, können unterschiedliche Ansätze erfordern.

Abschnitt 1.5 zu den Themen Teststufen und Projektarten sowie die Anhänge A »Softwarequalitätsmerkmale« und B »Last- und Performanztest« bieten eine Einführung in dieses Thema und sind als Exkurs, d.h. nicht als Bestandteil des Lehrplans zum ISTQB® Certified Tester Advanced Level Testautomatisierungsentwickler zu verstehen.

1.4.2 Testautomatisierungsarchitektur

Die Architektur der Testautomatisierungslösung ist entscheidend für die Akzeptanz der Lösung sowie deren Bestand und Nutzung auf lange Sicht. Der Entwurf einer geeigneten Testautomatisierungsarchitektur (TAA) ist auch ein Kernthema der Ausbildung zum Testautomatisierungsentwickler. Es erfordert auch eine gewisse Erfahrung, um die Anforderungen an die Architektur optimal umzusetzen. Aus diesem Grund gibt es in vielen Testautomatisierungsprojekten einen Testautomatisierungsarchitekten, ähnlich dem Softwarearchitekten in der Entwicklung, der das Projekt zumindest zu Beginn und bei größeren Anpassungen begleitet.

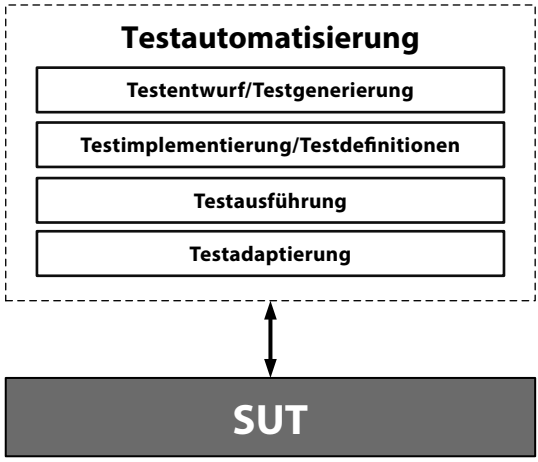

Abb. 1–4
Einfache, schematische Darstellung der Schichten einer generischen Testautomatisierungsarchitektur

Anforderungen an die Testautomatisierungsarchitektur

- Die Architektur der Testautomatisierungslösung ist eng mit der **Architektur des SUT** verbunden. Die einzelnen Komponenten, Benutzeroberflächen, Dialoge, Schnittstellen, Services und technischen Konzepte, verwendeten Sprachen etc. müssen adressiert sein.
- Bereits in der Test- und Testautomatisierungsstrategie sollte klar definiert werden, welche **funktionalen und nicht funktionalen Anforderungen des SUT** durch die Testautomatisierung – und damit durch die Testautomatisierungsarchitektur – adressiert und unterstützt werden sollen. Dies werden in der Regel die wichtigsten Anforderungen an das Softwareprodukt sein. Anhang A gibt einen Überblick über die Softwarequalitätsmerkmale nach ISO 25010 (Teil der Normenreihe ISO/IEC 25000:2014).
- Aber auch die **funktionalen und nicht funktionalen Anforderungen der Testautomatisierungslösung** müssen angemessen berücksichtigt werden. Insbesondere die Anforderungen hinsichtlich Wartbarkeit, Performanz und Erlernbarkeit stehen beim Entwurf einer Testautomatisierungsarchitektur im Vordergrund. Das SUT wird kontinuierlich weiterentwickelt. Modifizierbarkeit und Erweiterbarkeit müssen daher in hohem Maße gegeben sein. Modulare Konzepte oder die Trennung von funktionaler und technischer Implementierungsschicht sind Möglichkeiten, um dies zu gewährleisten. Mit zunehmendem Umfang der automatisierten Testsuite wird die Performanz der Testautomatisierungslösung ein immer wichtigeres Thema. Verstärktes Testen über die API-Schnittstellen statt über die GUI kann zu erheblichen Effizienzsteigerungen führen. Auch sollte die Testautomatisie-

rungslösung nicht eine Geheimwissenschaft für einige wenige Experten sein. Daher sind Verständlichkeit und Erlernbarkeit ebenfalls sehr wichtig. Es lohnt sich auch, die Qualitätsmerkmale in Anhang A zu betrachten und sie für die Anwendung auf die Testautomatisierungsarchitektur zu bewerten.

Um die bestmögliche Architektur für die Testautomatisierungslösung zu entwickeln, ist die Zusammenarbeit mit den Softwareentwicklern und -architekten sehr hilfreich, wenn nicht sogar unerlässlich. Denn um die oben genannten Anforderungen zu erfüllen, ist ein tiefes Verständnis der SUT-Architektur erforderlich.

1.4.3 Testbarkeit des SUT

Natürlich stellt auch die Testbarkeit, genauer gesagt, die automatisierte Testbarkeit des SUT einen wesentlichen Erfolgsfaktor dar. Die Testautomatisierungswerkzeuge müssen Zugriff auf die Objekte und Elemente der diversen Benutzer- oder Systemschnittstellen haben oder auf Komponenten und Services der Systemarchitektur, um diese zu identifizieren und ansteuern zu können.

Testautomatisierungswerkzeuge stellen dafür eine Reihe von Adaptern für die Automatisierung auf Basis unterschiedlichster Technologien und Plattformen bereit. Ob .NET, Java, SAP, Web, Desktop oder mobile Lösung, Windows, Linux, Android oder iOS, Google Chrom, Internet Explorer, Microsoft Edge, Mozilla Firefox oder Safari – die Palette ist riesig.

Die Hersteller richten ihre Lösungen an den gängigen Standards dieser Technologien und Plattformen aus. Problematisch wird es dann oft, wenn das SUT davon abweichende Implementierungen und Konzepte enthält. Daher ist es auch notwendig, im Zuge eines Proof of Concept die prinzipielle Automatisierbarkeit des SUT festzustellen und auch die passendste Automatisierungslösung zu finden. Denn dreierlei ist schwierig oder teuer: Den Hersteller eines Automatisierungswerkzeugs dazu zu bewegen, sein Produkt nach Ihren Vorstellungen umzubauen; die Entwicklungsabteilung davon zu überzeugen, die Architektur des SUT anzupassen und selbst entwickelte Klassenbibliotheken gegen andere auszutauschen; oder in der Testautomatisierungslösung mit aufwendigen Konstrukten irgendwie eine Lösung herbeizuführen.

Aber mit zunehmender Verbreitung der Testautomatisierung im Unternehmen kann, insbesondere in agilen Entwicklungsszenarien, die Automatisierbarkeit der Testdurchführungen als neues Qualitätsmerkmal für Softwareapplikationen an Bedeutung gewinnen.

Für den automatisieren Test über die GUI sollten etwa die Elemente und Daten für die Interaktion mit der GUI so weit wie möglich von ihrem Layout entkoppelt werden. Für den API-Test können entsprechende Schnittstellen (von Klassen, Modulen/Komponenten etc. oder der Kommandozeile) öffentlich bereitgestellt oder zusätzlich entwickelt werden.

Für jedes SUT gibt es Bereiche (Klassen, Module, funktionale Einheiten), die leichter zu automatisieren sind, und welche, wo die Automatisierung sehr aufwendig werden kann. Mögliche Showstopper sollten bereits in der Evaluierung und Auswahl eines Werkzeugs adressiert werden. Daher sollte man sich zu Beginn eher auf gut automatisierbare Testbereiche fokussieren, denn ein wichtiger Erfolgsfaktor ist die möglichst einfache Implementierung und Verteilung automatisierter Testskripte. Der Nachweis erfolgreicher, automatisierter Testdurchführungen hilft dem Projekt und unterstützt die Investition in den Ausbau der Testdurchführung. Versteigt man sich jedoch zu sehr in kritische Bereiche, liefert man unter Umständen kaum Ergebnisse und Mehrwert für das Vorhaben.

1.4.4 Testautomatisierungsframework

Ein Testautomatisierungsframework (TAF) muss einfach verwendbar, gut dokumentiert und vor allem gut wartbar sein. Die Grundlage dafür wird in der Testautomatisierungsarchitektur gelegt. Das Testautomatisierungsframework soll auch einen konsistenten Ansatz zur Testautomatisierung gewährleisten.

Dabei sind folgende Faktoren besonders wichtig:

- **Implementierung von Berichtsfunktionen**
  Die Testberichte müssen Informationen über die Qualität des SUT liefern (bestanden/nicht bestanden/fehlerhaft/nicht ausgeführt/abgebrochen, statistische Daten usw.). Die Berichte müssen diese Informationen in angemessener Form für die verschiedenen Beteiligten (Tester, Testmanager, Entwickler, Projektleiter und andere Beteiligte) darstellen, um einen Überblick über die Qualität des SUT zu erhalten.
- **Unterstützung bei der einfachen Fehlersuche und Fehlerbeseitigung**
  Zusätzlich zur Ausführung und Protokollierung von Tests soll das Testautomatisierungsframework eine einfache Möglichkeit zur Fehlersuche bei fehlgeschlagenen Tests bieten. Dabei können folgende Gründe für das Auftreten von Fehlern verantwortlich sein und idealerweise kann das Framework eine entsprechende Klassifizierung vornehmen bzw. diese in der Fehleranalyse unterstützen:

- Fehler im SUT
- Fehler in der Testautomatisierungslösung (TAS)
- Probleme mit den Testskripten
- Probleme mit Testumgebung (z.B. nicht funktionierende Services, fehlende Testdaten o.Ä.)

- **Korrekte Einrichtung der Testumgebung**
  Für die automatisierte Testausführung ist eine dedizierte Testumgebung erforderlich, die die verschiedenen Testwerkzeuge konsistent integriert. Wenn die automatisierte Testumgebung oder die verwendeten Testdaten keinen Eingriff oder keine Konfiguration zulassen, können die Testskripte möglicherweise nicht entsprechend den Anforderungen für die Testausführung eingerichtet und durchgeführt werden. Dies wiederum kann zu unzuverlässigen, irreführenden oder sogar falschen Testergebnissen (falsch positive oder falsch negative Ergebnisse) führen. Ein falsch positives Testergebnis zeigt an, dass ein Problem vorliegt (d.h., der automatisierte Test schlägt fehl), auch wenn kein Fehler vorhanden ist. Ein falsch negatives Testergebnis liegt vor, wenn ein Test erfolgreich ist (der automatisierte Test also auf keinen Fehler aufläuft), obwohl das System fehlerhaft ist.
- **Dokumentation der automatisierten Testfälle**
  Die Ziele der Testautomatisierung müssen klar definiert und beschrieben sein. Welche Teile der Software sollen in welchem Umfang getestet werden? Welcher Testautomatisierungsansatz soll verwendet werden? Welche (funktionalen und nicht funktionalen) Eigenschaften des SUT sollen durch die Automatisierung geprüft werden? Und aus der Dokumentation der automatisierten Testfälle oder Testfallsets muss erkennbar sein, welches Testziel durch sie verfolgt wird.
- **Rückverfolgbarkeit der automatisierten Tests**
  Nicht selten findet man automatisierte Testsuiten, in denen nicht oder kaum noch nachvollziehbar ist, welche Fachlichkeit, welches fachliche Testszenario abgedeckt wird. Und weil man sich nicht sicher ist, implementiert man neue Testskripte. Neben dem grundsätzlichen Problem der Intransparenz entstehen dadurch eine Menge unnötiger Redundanzen und Unsicherheiten. Deshalb muss das Testautomatisierungsframework auch die Nachvollziehbarkeit der automatisierten Testfallschritte zu den Testfällen unterstützen.
- **Einfache Wartung**
  Eines der größten Risiken für den Erfolg eines Testautomatisierungsprojekts ist der Wartungsaufwand. Im Idealfall sollte der Aufwand für die Pflege bestehender Testskripte nur einen kleinen Prozentsatz

des Testautomatisierungsaufwands ausmachen. Außerdem sollte der Aufwand für die Anpassung der Testautomatisierungslösung in einem gesunden Verhältnis zum Umfang der Änderungen am SUT stehen. Wenn die Testautomatisierung anfängt, teurer zu werden als die Entwicklung des SUT, wird das Ziel der Kostensenkung durch den Einsatz der Testautomatisierung wahrscheinlich nicht erreicht werden. Daher müssen die automatisierten Testfälle einfach zu analysieren, zu ändern und zu erweitern sein. Eine gut durchdachte, an das SUT angepasste Modularisierung erlaubt einen hohen Grad der Wiederverwendung einzelner Komponenten und reduziert so die Anzahl der im Bedarfsfall anzupassenden Artefakte.

- **Aktualität der automatisierten Testfälle**
  Es ist nicht ungewöhnlich, dass automatisierte Testfälle fehlschlagen, nicht weil ein Anwendungsfehler entdeckt wurde, sondern weil in der Zwischenzeit Änderungen an den fachlichen oder technischen Spezifikationen vorgenommen wurden, die in den Testskripten noch nicht nachgezogen wurden. Natürlich sollten die betroffenen Testfälle nicht einfach verworfen, sondern entsprechend angepasst werden. Besser ist es jedoch, sicherzustellen, dass der Testautomatisierungsentwickler alle Informationen über Änderungen am SUT durch geeignete Prozesse, Dokumentationen und Werkzeuge erhält, um die Testsuite rechtzeitig zu aktualisieren.
- **Planung der Softwareverteilung**
  Das Testautomatisierungsframework sollte auch das Versions- und Konfigurationsmanagement der Testautomatisierungslösung, die mit den Versionen des SUT synchron gehalten werden muss, durch den geeigneten Einsatz von Werkzeugen und Standards unterstützen. Die Verteilung, Änderung und Neuverteilung der Testskripte müssen so einfach wie möglich sein.
- **Außerbetriebnahme von automatisierten Tests**
  Wenn bestimmte automatisierte Testsequenzen nicht mehr benötigt werden, unterstützt das Testautomatisierungsframework deren Außerbetriebnahme. In den meisten Fällen reicht es nicht aus, ein oder mehrere Skripte einfach zu löschen. Es muss auch möglich sein, alle Abhängigkeiten zu diesen Komponenten einfach zu bearbeiten, um die Konsistenz der Testautomatisierungslösung zu erhalten. Toter Code sollte – wie bei der Softwareentwicklung – vermieden werden.
- **Überwachung und Wiederherstellung des SUT**
  Normalerweise muss das SUT laufend auf die kontinuierliche Ausführung eines oder mehrerer Testfälle überwacht werden. Wenn ein schwerer Fehler (z.B. ein Absturz) im SUT auftritt, muss das Test-

automatisierungsframework in der Lage sein, den aktuellen Testfall zu überspringen, das SUT in einen konsistenten Zustand zurückzuversetzen und mit der Ausführung des nächsten Testfalls fortzufahren.

Wartung des Testautomatisierungscodes

Der Testautomatisierungscode kann oft den Umfang des Entwicklungscodes erreichen und zudem recht komplex sein. Dies ist insbesondere dann der Fall, wenn komplizierte Testsequenzen innerhalb eines Testskripts abgebildet werden oder wenn eine spezifische Handhabung von technischen Schnittstellen oder Oberflächenelementen erforderlich ist. Möglicherweise sind auch zeitliche Trigger oder Verzögerungen zu implementieren oder es werden Testfallketten implementiert, die über (Zwischen-)Ergebnisdaten miteinander verknüpft sind. Dadurch wird die damit verbundene Wartung komplex und der Wartungsaufwand sehr hoch. Hinzu kommen oft die verschiedenen verwendeten Testwerkzeuge, unterschiedliche Arten der Verifikation und Validierung und die diversen zu pflegenden Testmittel (z.B. Testeingabedaten, Testorakel, Testberichte). Was für die Testautomatisierungsarchitektur gilt, gilt unbedingt auch für den entwickelten Testcode bzw. die Testskripte: Der Wartbarkeit ist höchste Aufmerksamkeit zu schenken!

Empfehlungen zur Verringerung des Wartungsaufwands:

- **Technische Unabhängigkeit**
  Es muss darauf geachtet werden, technische Abhängigkeiten und Verbindungen zum SUT zu vermeiden oder zu minimieren. Aus diesem Grund werden in den verschiedenen Frameworks und Automatisierungswerkzeugen die konkreten technischen Anbindungen an die grafische GUI oder an API-Schnittstellen in eine separate, zentrale Schicht verlagert. Die Trennung von technischer und fachlicher Sicht ist essenziell und darf vom Testautomatisierungsentwickler nicht ausgehebelt werden.
- **Datenunabhängigkeit**
  Was für die technische Anbindung an das SUT gilt, gilt auch für die in der Testautomatisierung relevanten Stamm-, Bewegungs- und Steuerdaten. Diese sollten ebenfalls in eine Datenzugriffsschicht abstrahiert werden. Hartcodierte Testdaten in den Testskripten sollten vermieden werden, auch z.B. bei Verifikationen. Datenänderungen, wie z.B. ein neuer Steuersatz, eine geänderte Bestätigungsmeldung usw., sollten nicht zur Überarbeitung vieler Skripte führen. Das Risiko von Änderungen sollte auch dann berücksichtigt werden,

wenn eine Abhängigkeit von Testskripten untereinander über deren Ein- und Ausgabedaten besteht.

- **Umgebungsunabhängigkeit**
  Der implementierte Satz von automatisierten Testfällen sollte auch in verschiedenen Testumgebungen und auf verschiedenen Plattformen ausführbar sein. Einstellungen oder Daten, die in der Automatisierung benötigt und von der Betriebsplattform übernommen werden, wie z.B. Systemzeit oder Lokalisierungsparameter des Betriebssystems oder Daten aus anderen Anwendungen der Testumgebung, sollten mithilfe von Platzhaltern oder Konfigurationsdateien und -einstellungen implementiert werden. Viele dieser Aspekte sollten durch das Testautomatisierungsframework bereitgestellt und genutzt werden.
- **Dokumentation**
  Eine gute (Inline-)Dokumentation der erstellten Testskripte hilft enorm bei Anpassungen und Erweiterungen der Testskripte und bei der Fehleranalyse. Die Einführung geeigneter Entwicklungs- und Dokumentationskonventionen, die der Lesbarkeit und Nachvollziehbarkeit dienlich sind, trägt wesentlich dazu bei, den Wartungsaufwand zu reduzieren.

1.5 Exkurs: Teststufen und Projektarten

Die Festlegung einer Testautomatisierungsstrategie, die Ausarbeitung einer Testautomatisierungsarchitektur und die Entwicklung eines Testautomatisierungsframeworks geschieht nicht kontextfrei. Die Automatisierung der Testaktivitäten erfolgt auf unterschiedlichen Ebenen des Entwicklungsprozesses bzw. Teststufen. Und je nach Projekttyp können sich der strategische, methodische und technische Zugang zur Testautomatisierung ebenfalls unterscheiden. Die folgenden Kapitel geben Hinweise und Ideen zur Ausgestaltung einer passenden Strategie, einer Architektur und eines Frameworks.

1.5.1 Testautomatisierung auf unterschiedlichen Teststufen

Es gibt viele Modelle, an denen sich Softwareentwicklungsprozesse orientieren können. Ein weitverbreitetes Modell ist das allgemeine V-Modell. Es bildet eine Grundlage für die Klassifizierung von Aktivitäten und deren Abhängigkeiten. Im Test sind die verschiedenen Teststufen an diesem Modell angelehnt, dabei spielt die Automatisierung je nach Teststufe eine unterschiedliche Rolle.

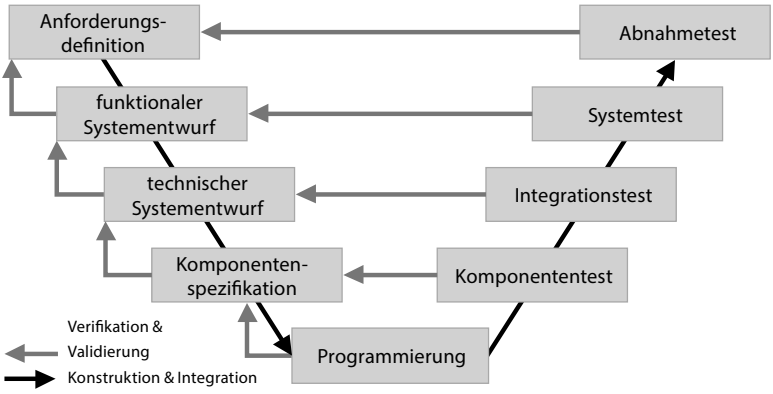

Abb. 1–5
Allgemeines V-Modell

Komponententest

Typische Beispiele für den automatisierten Test sind Komponenten-, Modul- oder auch Unit Tests. Diese Tests fallen üblicherweise in den Verantwortungsbereich des Entwicklungsteams und werden somit oft als »automatisierte Entwicklertests« bezeichnet.

Der Komponententest zielt darauf ab, die Funktionalität in der kleinsten Einheit zu verifizieren, in der Software sinnvoll getestet werden kann. Übliche Definitionen für diese kleinste Einheit sind Klassen, Funktionen, Methoden oder Prozeduren, aber je nach Sprachparadigma sind auch andere Definitionen möglich.

Da das zu testende System aller Wahrscheinlichkeit nach nicht nur aus solchen eigenständigen Einheiten besteht, sondern auch Abhängigkeiten zwischen diesen Einheiten existieren, muss ein Testrahmen geschaffen werden, in dem diese Einheiten isoliert ausführbar und steuerbar gemacht werden können.

Mocks

In objektorientierten Umgebungen erfolgt dies durch eine Ersetzung der Abhängigkeiten mit möglichst einfachen »Mocks«, die im Gegensatz zur »echten« Abhängigkeit so wenig Funktionalität wie möglich besitzen. Üblicherweise bieten Mocks die Möglichkeit, zu spezifizieren, welche Werte sie bei Aufrufen zurückgeben, und zu überprüfen, ob und wie oft Methoden aufgerufen wurden.

Auf diese Weise werden die kleinstmöglichen Testobjekte isoliert testbar gemacht. Der automatisierte Komponententest eignet sich speziell dafür, dem Entwicklungsteam rasch Feedback über Auswirkungen von Änderungen am Testobjekt zu geben, und bietet damit Sicherheit, bei größeren Änderungen innerhalb einer Einheit (z.B. Refactoring) jederzeit feststellen zu können, ob bestehende Funktionalität beeinträchtigt wurde.

Auch die Robustheit der einzelnen Komponenten ist auf dieser Teststufe gut testbar, da auf die Komponenten meist ohne die Einschränkung von vorgelagerten Datenvalidierungen zugegriffen werden kann.

Test Driven Development

Der Komponententest wird immer mehr im Sinne von »Test Driven Development« angewendet. Test Driven Development (TDD) ist ein zentraler Bestandteil vieler agiler Methoden in der Softwareentwicklung, z. B. von Extreme Programming (XP). Die Idee von TDD ist, dass das Schreiben von Code durch den Test gesteuert wird. Üblicherweise werden Tests nach oder parallel zur Implementierung geschrieben, um den Code zu testen. Das hat den Nachteil, dass eine hohe Testüberdeckung durch automatisierte Tests nur mit hohem Aufwand möglich ist.

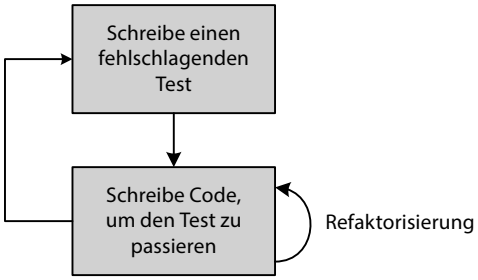

Abb. 1–6
Vorgehensweise bei Test Driven Development

TDD geht einen neuen Weg, der ein radikales Umdenken erfordert: zuerst der Test und danach die Codierung. Es wird jeweils nur so viel neuer Code geschrieben, wie es der automatisiert ausgeführte Test verlangt, um keinen Fehler zu melden. Der Code ist möglichst einfach und verständlich zu gestalten. Der Vorteil liegt auf der Hand: Es gibt zu jedem Zeitpunkt im Entwicklungszyklus ein Set an automatisierten Tests, die den aktuell vorliegenden Code in seiner Gesamtheit testen.

Folgende Schritte werden bei Verwendung dieser Methode beachtet:

1. Erstellung der benötigten Klasse
2. Erstellung einer Testklasse für die Klasse
3. Definition und Anlegen der Methoden der Klasse sowie der Testklasse

4. Implementierung der Tests in die Methoden der Testklasse
   a) Definition der Eingabewerte
   b) Definition der erwarteten Ergebnisse
   c) Überprüfung durch Assertions auf Richtigkeit und Fehlerfall (Extremfall)
5. Implementierung der Logik in die Methoden der Klasse

Für einen erfolgreichen Einsatz von TDD ist eine Toolunterstützung (Frameworks) auf den Ebenen Testentwicklung, Testautomatisierung und Build-Automatisierung unbedingt notwendig.

Integrationstest

Der Integrationstest beschreibt den expliziten Test des Zusammenspiels mehrerer Komponenten. Auf dieser Teststufe können, je nach Situation, sowohl Methoden des Komponententests als auch des Systemtests angewandt werden.

- **Komponentenintegrationstest**
  Beim Komponentenintegrationstest wird die zu testende Komponente nicht wie im Komponententest durch Techniken wie Mocking isoliert, sondern ihr Zusammenspiel mit den entsprechenden Komponenten getestet. Dieser Test ist in den meisten Fällen vollautomatisiert und bietet unter anderem Sicherheit bei größeren Refactorings, die auch die Interaktion von mehreren Komponenten betreffen.
- **Subsystem- und Systemintegrationstest**
  Beim (Sub-)Systemintegrationstest wird ähnlich vorgegangen: Teile des Gesamtsystems werden miteinander integriert und ihre Interaktion auf Korrektheit überprüft. Hierfür können Simulatoren und Testrahmen notwendig sein. In diesem Fall sind die zu testenden Komponenten üblicherweise keine Klassen und Module, sondern bereits Pakete solcher Einheiten oder Teilsysteme.

Robustheit und Datenintegrität

Der Integrationstest ist die Teststufe der Wahl, um Robustheit und Datenintegrität zwischen Komponenten sowie die Einhaltung von Protokollen und vereinbarten Verwendungsweisen zu überprüfen.

Auch auf dieser Ebene ist in vielen Fällen zumindest eine Teilautomatisierung sinnvoll, da ein nur teilweise integriertes System häufig noch keine Benutzerschnittstelle hat bzw. diese noch nicht integriert ist.

Systemtest

In sequenziellen Entwicklungsmodellen findet der funktionale Systemtest in seiner Gesamtheit an einer zentralen Stelle im Entwicklungsprozess statt. Dennoch gibt es Szenarien, in denen ein Regressionstest notwendig wird:

- Nachträgliche Change Requests und Erweiterungen
- Fehlerkorrekturen
- Refactoring
- Redesign
- Wartungsarbeiten

Wenn im Test von Testautomatisierung die Rede ist, so ist zumeist der automatisierte Systemtest gemeint. Dieser setzt in den meisten Tools an der grafischen Benutzerschnittstelle und der Datenbank an, um durch Tester spezifizierte und zumeist zuvor manuell durchgeführte Testfälle automatisiert abzuarbeiten. Dies ist mit ein Grund, warum der automatisierte Systemtest eine der aufwendigsten Varianten von Testautomatisierung sein kann.

- Im Fokus ist das gesamte zu testende System – unter Umständen inklusive weiterer, dahinter liegender Systeme.
- Die Testfälle erfordern fachliches Verständnis.
- Die betroffenen Schnittstellen sind für die Interaktion mit einem Benutzer, nicht mit einem Programm konzipiert.
- Für die Vorbereitung von ausreichenden Testdaten können keine Mocks verwendet werden – dies muss im System selbst erfolgen.
- Testtreibererstellung für die Simulation von Drittsystemen ist erforderlich, wenn kein Systemintegrationstest gewünscht ist.

Auch bei nicht funktionalen Systemtests ist Automatisierung in vielen Fällen ein essenzielles Werkzeug. Last- und Performanztests sind ohne Automatisierung erst gar nicht vernünftig machbar.

Abnahmetest

In klassischen, sequenziellen Entwicklungsmodellen steht der Abnahmetest am Ende eines Softwareentwicklungsprozesses, nach dem Systemtest. Hier wird die erstellte Software anhand der zu Beginn aufgestellten Anforderungsdokumentation vom Kunden abgenommen.

Agile und iterative Ansätze

In einigen Softwareentwicklungsmodellen, speziell in agilen und iterativen Ausprägungen, werden im Vorhinein unter Einbezug aller Disziplinen konkrete Akzeptanzkriterien in Form von Testfällen festgelegt und formuliert, die als Basis für die Feststellung der Komplettierung einer Funktionalität dienen.

Behaviour Driven Development

Eine Technik, die als Weiterführung von Test Driven Development gesehen werden kann und eine automatisierte Überprüfung der Erfüllung dieser Akzeptanzkriterien beinhaltet, ist Behaviour Driven Development (BDD) [URL: BDD]. In BDD sollen Testfälle in einer Form festgelegt werden, die einerseits fachlich verständlich und andererseits automatisiert durchführbar ist. Dies dient dazu, dem Entwickler ausreichend Hintergrundinformationen über den Zweck des erwarteten Codes in Form von Beispielen zu vermitteln.

Domänenspezifische Sprachen

Hierfür werden domänenspezifische Sprachen definiert, in denen Testfälle festgehalten werden können. Die Automatisierung dieser Testfälle bzw. die Erstellung einer Automatisierungsumgebung, die diese Testfälle automatisiert abarbeiten kann, ist Teil der Entwicklungsarbeit und kann im Vorfeld und unter Zuhilfenahme von Frameworks durchgeführt werden.

Ein klassisches Schema in BDD ist die Form, in der Testfälle beschrieben werden: Diese bestehen aus Vorbedingungen, Aktionen und Überprüfungen.

Ein Beispiel:

- **Gegeben** ein Kunde unter 16 Jahren ist angemeldet
- Und ein leerer Warenkorb des Kunden
- Und eine Veranstaltung mit Altersbeschränkung ab 16
- **Wenn** der Kunde die Veranstaltung in den Warenkorb legt
- **Dann** soll eine Fehlermeldung »Die Veranstaltung hat eine Altersbeschränkung von 16 Jahren« erscheinen ...
- ... und kein Ticket im Warenkorb sein

In diesem Beispiel beschreibt der erste Abschnitt die erforderlichen Testdaten, die vor Durchführung des Testfalls zur Verfügung gestellt werden müssen. Der zweite Abschnitt beschreibt eine Aktion auf dem Testobjekt. Der dritte Abschnitt legt Überprüfungen der Reaktion des zu testenden Systems und die dazugehörigen erwarteten Ergebnisse fest.

1.5.2 Einsatzgebiet nach Projektart

Unterschiedliche Projektarten bedingen unterschiedliche Szenarien der Automatisierung.

Unterschiedliche Projekttypen verlangen einen differenzierten Ansatz in der Testautomatisierung. Der automatisierte Test einer Einzelplatzanwendung hat unter Umständen einen eingeschränkten technologischen Umfang und den Schwerpunkt auf der funktionalen Korrektheit. Darüber hinaus steht der Regressionstest mehrerer geplanter Releases im Vordergrund. Demgegenüber ist der Einsatz von Testautomatisierung in einem Datenmigrationsprojekt normalerweise eher als Konsistenz- und Vergleichstest zu verstehen und nicht als Wiederholungsfall auf Jahre hinaus zu konzipieren.

Das klassische Softwareentwicklungsprojekt

So einfach dieser Projekttyp aufgrund der zumindest theoretisch geforderten, klaren fachlichen Vorgaben und Dokumentationen für den Softwaretest scheint, so schwierig ist dieser für die Implementierung einer umfassenden Testautomatisierung. Der Grund dafür liegt jedoch nicht in den fachlichen oder technischen Herausforderungen. Im Gegenteil, die fachlichen Anforderungen und die dazugehörigen Testfälle können meist erschöpfend und strukturiert erarbeitet und rechtzeitig zur Testdurchführung bereitgestellt werden. Die große Problematik für die Automatisierung besteht in der zeitgerechten Verfügbarkeit der zu automatisierenden Applikation. Mit der oft schon verzögerten Bereitstellung des Testobjekts wird bereits die Testdurchführung und ein rasches Feedback für die Entwicklung erwartet und nicht der Start einer unter Umständen mühsamen Automatisierung.

Daher steht in klassischen Softwareentwicklungsprojekten vorerst der manuelle Test im Vordergrund und nur in jenen Fällen, wo ein langdauerndes, über mehrere Lieferversionen geplantes Projekt vorliegt oder wo bereits in Voraussicht auf eine langjährige Produktweiterentwicklung geplant wird, hält die Testautomatisierung Einzug in das Vorhaben. Punktuell wird aber auch auf die Automatisierung zurückgegriffen, wenn der Test stark datengetrieben ist und es z.B. gilt, eine Vielzahl von Wertekombinationen zu testen. Hier rechnet sich die Investition in die Automatisierung unter Umständen schon mit einer einzigen Testdurchführung.

Das Wartungsprojekt und die Produktweiterentwicklung

Gegenwärtig ist die Automatisierung von Softwaretests in diesen Projekten nicht sehr verbreitet. Dies gilt aber meist auch für die geordnete Testdurchführung als Ganzes. Der Test von kleineren Erweiterungen und Anpassungen wird durch die Entwickler selbst durchgeführt oder an die Fachbereiche delegiert. Dieses Szenario ist jedoch in zunehmendem Maße schwer zu leben. Die Applikationen werden immer komplexer und die benötigten Testressourcen immer weniger verfügbar bzw. deren Kosten immer transparenter. Wenn z.B. die Kosten der Fachbereiche nachvollziehbar und hoch sind, wird es immer schwieriger, Argumente für den Ansatz des Testens durch den Fachbereich zu finden.

Daher stellt dieser Projekttyp aus organisatorischer Sicht einen idealen Ausgangspunkt für die Testautomatisierung dar. Nach der grundlegenden Definition einer Automatisierungsstrategie kann diese laufend und in kleinen Schritten umgesetzt werden. Diese Schritte erlauben einen stetigen Lern- und Adaptionszyklus. Die Umsetzungen können z.B. entlang folgender Nutzenüberlegungen priorisiert werden:

Schritte zur Automatisierung in Wartungsprojekten

- Automatisierung der Testfälle, die die aktuellen Änderungen betreffen. Neu entwickelte und von Veränderungen betroffene Bereiche sind wesentlich fehleranfälliger als Bereiche der Applikation, die bereits lange in Produktion sind. Das gilt auch noch für mehrere Folgereleases.
- Automatisierung jener Tests, die normalerweise vom Fachbereich durchgeführt werden. In einem ersten Ansatz werden diese Tests auch auf einem vergleichbaren Qualitäts- und Detaillevel umgesetzt. Dies reduziert den Aufwand in den Fachbereichen und schafft somit eine massive Kosteneinsparung bei zumindest gleichbleibender Qualität.
- Automatisierung von Testfällen, die aus der Analyse von Problemen und Fehlern während des Betriebs abgeleitet werden. Diese Quelle eignet sich besonders für die Auswahl der zu automatisierenden Testszenarien, insbesondere im Hinblick auf die Stabilisierung des SUT.
- Laufende Optimierung und Ausbau der oben angeführten Szenarien, vor allem der Regressionstests, wie sie normalerweise durch den Fachbereich oder eine Testabteilung manuell durchzuführen sind.

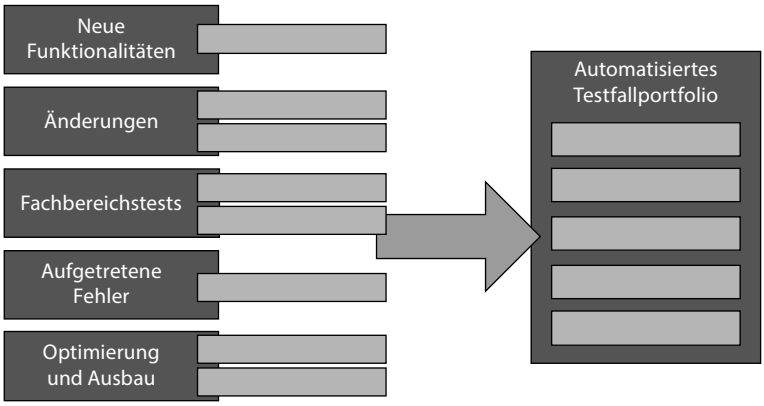

Abb. 1–7
Quellen für Änderungen im automatisierten Testfallportfolio in Wartungsprojekten

Das SAP-Projekt

Ein Sonderfall für ein Implementierungs- oder Weiterentwicklungsprojekt sind die vielen SAP-Implementierungen. Die laufenden Releasewechsel, Upgrades oder Enhancement Packages stellen eine große Herausforderung an die jeweilige Organisation dar. Jedes Mal sind insbesondere die Mitarbeiter aus den Fachbereichen aufgerufen, nach den durchgeführten Änderungen die Funktionstüchtigkeit des Systems zu überprüfen. Speziell davon betroffen sind alle Customizing-Einstellungen, individuelle Erweiterungen und die Systemschnittstellen. Der dafür erforderliche Testaufwand erreicht schnell ein fast unüberschaubares Maß, was einer der Gründe dafür ist, dass einige Unternehmen nicht jedes Upgrade oder Package installieren und lieber auf Systemverbesserungen verzichten, als das stabile Systemverhalten zu gefährden.

Automatisierung im SAP-Umfeld

Es ist daher sehr verständlich, dass der Ruf nach automatisierten Tests gerade in diesem Umfeld immer lauter wird. Die Ausgangslage dafür ist sehr gut: eine hoch standardisierte Anwendungslandschaft und weitgehend einheitliche Benutzeroberflächen und Schnittstellenarchitektur. Viele Hersteller kommerzieller Automatisierungslösungen sind zudem SAP-zertifiziert und bedienen sich direkt der transparenten SAP-Technologien.

Die Gründe, warum die Automatisierung im SAP-Umfeld immer noch nicht Standard ist, liegen zum einen in den Problemen der Testumgebung bzw. der Testdatenbereitstellung und Wiederherstellung einer konsistenten Datenbasis für die systemübergreifende Testdurchführung. Zum anderen liegt es aber auch an der nicht einfach zu beantwortenden Frage, welche der Tausenden möglichen Tests tatsäch-

lich automatisiert werden sollen. Die richtigen Antworten auf diese konzeptionellen Fragen sind jedoch der Schlüssel zu einer erfolgreichen Automatisierung in SAP-Projekten.

Agile Projekte

Hocheffiziente Testmethoden und Pair Testing

»Working software is the primary measure of progress«, so heißt es in einem der zwölf Prinzipien des Agilen Manifests [URL: AGILE]. Das Ziel jedes Sprints ist daher, dass an dessen Ende eine funktionstüchtige, potenziell auslieferbare und daher korrekte Software verfügbar ist. Insbesondere bei sehr kurzen Entwicklungszyklen ist dies jedoch schwer zu gewährleisten, wenn gleichzeitig bis zum letzten Tag programmiert wird. Um dennoch sicherzustellen, dass alle notwendigen Testaktivitäten durchgeführt werden können, ist Testautomatisierung besonders im agilen Umfeld unumgänglich. Die Automatisierung wird jedoch dadurch erschwert, dass häufig keine stabilen Testobjekte zur Verfügung stehen, vielmehr sind diese Artefakte ständigen Veränderungen unterworfen. Diese Aussage trifft nicht nur auf jene Applikationsteile zu, die während eines Sprints umgesetzt werden, sondern auch auf solche aus vorhergehenden Iterationen. »Welcome changing requirements, even late in development«, lautet das entsprechende Prinzip im Agilen Manifest.

Für die Qualitätssicherung und vor allem für die Testautomatisierung bedeutet dies, dass nicht nur sehr oft vollständige Regressionstests durchgeführt werden müssen, sondern dass dieser Test kontinuierlich an neue Anforderungen sowie technische oder funktionale Änderungen angepasst werden muss. Daher ist es auch nicht überraschend, dass in vielen agilen Projekten mit gänzlich anderen Methoden und Ansätzen gearbeitet wird als in klassischen Projekten.

Wichtige Ansätze abseits von Testautomatisierung, die die tägliche Arbeit von agilen Testern und Entwicklern sehr stark prägen, sind der explorative Test und das Pair Testing (bzw. Pairing generell). Da diese Methoden jedoch nicht direkt durch Testautomatisierung unterstützt werden, seien interessierte Leser an dieser Stelle auf [Kaner et al. 11] bzw. [Baumgartner et al. 13] und [Linz 16] verwiesen.

Für den Einsatz von Testautomatisierung bedeutend sind jedoch die kontinuierliche Integration und kontinuierliche Auslieferung (Continuous Integration bzw. Continuous Delivery) sowie technische Ansätze wie testgetriebene Entwicklung (Test Driven Development) oder auch akzeptanztestgetriebene Entwicklung (Acceptance Test Driven Development).

Kontinuierliche Integration und Auslieferung

Kontinuierliche Auslieferung beschreibt im Grunde keine einzelne Technik oder Methode, sondern eine ganze Sammlung von Prinzipien mit dem Ziel, große Teile des Integrations- bzw. Auslieferungsprozesses zu automatisieren. Dazu gehören neben einer umfassenden Testautomatisierung (Komponententests, Systemtests, aber auch Abnahmetests) auch die kontinuierliche Integration, die automatisierte Bereitstellung von Testsystemen und die automatisierte Auslieferung auf unterschiedliche Systeme (sowohl Entwicklungs-, QA- als auch Produktivumgebung).

Martin Fowler fasst kontinuierliche Auslieferung wie folgt kurz zusammen [URL: Fowler]:

> *»You're doing continuous delivery when:*
>
> - *Your software is deployable throughout its lifecycle*
> - *Your team prioritizes keeping the software deployable over working on new features*
> - *Anybody can get fast, automated feedback on the production readiness of their systems any time somebody makes a change to them*
> - *You can perform push-button deployments of any version of the software to any environment on demand«*

Die kontinuierliche Integration stellt nur einen Teil eines kontinuierlichen Auslieferungsprozesses dar. Ein weiterer zentraler Bestandteil ist die automatisierte Durchführung von Tests auf den unterschiedlichen Teststufen. Daraus ergeben sich spezifische Anforderungen an die Testautomatisierungswerkzeuge :

- Kann die Testdurchführung in ein Build-System (z.B. Maven, Ant) integriert werden? Ist es also möglich, dass die automatisierten Testfälle bei jedem Build des Systems automatisch durchgeführt werden und abhängig vom Ergebnis der weitere Build-Verlauf (z.B. Build abbrechen bei fehlgeschlagener Testdurchführung) beeinflusst werden kann.
- Ist es möglich, die Testergebnisse von unterschiedlichen Teststufen einheitlich zu verwalten und darzustellen? Da vor allem in agilen Projekten auch die Testautomatisierung von Komponenten eine wichtige Rolle spielt, ist es essenziell, dass diese auf die gleiche Weise dargestellt und verwaltet werden können wie automatisierte Integrations- bzw. Systemtests.

- Wie lassen sich die automatisierten Testfälle auf unterschiedlichen Umgebungen durchführen? Diese Frage stellt sich speziell dann, wenn die Software tatsächlich kontinuierlich auch auf unterschiedliche Zielsysteme ausgeliefert werden soll. Dies sollte dann ohne Zusatzaufwand möglich sein – im Optimalfall lediglich durch die Änderung eines Konfigurationsparameters.

Zusammenfassend lässt sich sagen, dass die Automatisierung des funktionalen Regressionstests ein Muss für agile Projekte ist. Dies gilt sowohl für den Ausbau von Komponententests (speziell durch den Einsatz von testgetriebener Entwicklung) als auch für die Automatisierung von funktionalen Systemtests und automatisierten oder teilautomatisierten Abnahmetests. Damit dies möglich ist, dürfen die organisatorischen Rahmenbedingungen nicht außer Acht gelassen werden. In agilen Projekten bedeutet das, dass die Automatisierbarkeit bzw. die Testbarkeit, aber auch der Grad der Automatisierung für eine User Story als essenzielle Punkte in die Definition of Done des Teams aufgenommen werden müssen und wie jede andere Abnahmebedingung zu handhaben ist.

Automatisierung ist ein Muss für agile Projekte.

Dies ist besonders wichtig, um ein Bewusstsein für den mit jedem Sprint stetig steigenden Testaufwand zu schaffen. In der Praxis ist es zwar selten der Fall, dass bei jedem Sprint tatsächlich alle vorhandenen Funktionalitäten erneut getestet werden, aber selbst wenn nur ein Teil davon in den Regressionstest einbezogen wird, verschiebt sich das Verhältnis von zu testender zu entwickelnder Funktionalität innerhalb eines Sprints schnell zulasten des Tests. Das Team muss darauf reagieren, indem es die Automatisierung vorantreibt und die Sprint-Planung entsprechend anpasst. Ein gutes, agiles Team findet die richtige Lösung, denn das gesamte Team ist am Ende des Sprints für die funktionierende Software verantwortlich.

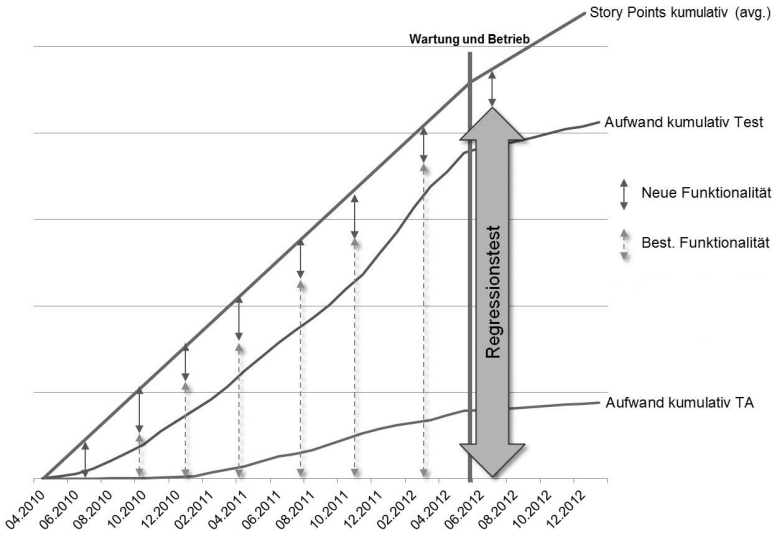

Abb. 1–8
Aufwandsentwicklung und kontinuierliches Wachstum des Regressionstestportfolios in agilen Projekten

DevOps

Neben »agilen Projekten« hat sich mit DevOps in den letzten Jahren eine weitere Idee in der Softwareentwicklung weit verbreitet und etabliert. Der Grundgedanke basiert dabei sehr stark auf agilen Prinzipien und Praktiken aus der agilen Softwareentwicklung, allerdings wird noch zusätzlicher Fokus auf die Integration der Fachbereiche und des Betriebs sowie der Automatisierung in sämtlichen Bereichen des Softwarelebenszyklus (Entwicklung, Test, Verteilung, Betrieb) gelegt. Auf abstrakter Ebene lässt sich die Zielsetzung von DevOps zu folgenden Zielen zusammenfassen:

- Verbesserung der Qualität des Gesamtsystems und damit Mehrwert für die tatsächlichen Endbenutzer
- Kürzere Lieferzyklen und dadurch effektivere Feedbackzyklen
- Kostenreduktion und Effizienzsteigerung
- Mehr Flexibilität und damit bessere Möglichkeiten, auf sich ändernde Rahmenbedingungen reagieren zu können

Ein Ansatz, um DevOps zu beschreiben, stellt das CALMS-Framework dar.

Das Akronym CALMS steht dabei für eine Reihe von Gesichtspunkten, unter denen eine Organisation betrachtet werden kann: Culture, Automation, Lean, Measurement und Sharing.

Culture

Eine essenzielle Voraussetzung ist die Tatsache, dass DevOps nicht lediglich als ein Prozess oder eine Herangehensweise an die Softwareentwicklung betrachtet werden kann. Vielmehr findet sich im Zentrum die Team- bzw. Organisationskultur, die stringent auf nahtlose, funktionsübergreifende Zusammenarbeit ausgerichtet ist.

In vielen Fällen können eine gemeinsame Vision und Mission des Teams hilfreich sein, um eine Grundlage für diese Kultur zu legen. Es gibt verschiedene Ansätze, diese zu entwickeln, aber Voraussetzung ist auf jeden Fall ausreichend Vertrauen (nicht nur das Vertrauen zwischen Teammitgliedern, sondern auch das Vertrauen des Managements in die Teammitglieder, DevOps und den damit verbundenen ROI).

Mit dieser offenen Kultur geht auch zwangsweise Transparenz in der Kommunikation einher. Sowohl Erfolge als auch Misserfolge sind Teil dieser Kultur und werden auf die eine oder andere Weise beachtet. Misserfolge sollen dazu führen, dass kontinuierliche Verbesserungen etabliert werden oder etwas Neues gelernt wurde. Gerade der Umgang mit Fehlern (»Fehlerkultur«) ist tief in der Organisationskultur verwurzelt und kann, zum Beispiel durch offene oder versteckte Schuldzuweisungen, den Erfolg von DevOps verhindern.

Automation

Grob gesagt sollten Organisationen bestrebt sein, so viele manuelle und wiederkehrende Aufgaben wie möglich zu automatisieren. Dabei muss allerdings auch stets auf Stabilität, Wartbarkeit und Einfachheit geachtet werden. Es gibt viele Teilbereiche, die unter dem Hauptthema der Automatisierung angesprochen werden können, und jeder für sich genommen ganze Artikel und Bücher füllen könnte. Daher sollen hier nur einige der wichtigsten zusammenfassend erwähnt werden:

- **Automatisierung des Build-Prozesses**
  Wie bereits zuvor erwähnt, ist die kontinuierliche Integration und Auslieferung hier eine Kernpraktik. Dazu gehört aber auch eine adäquate Verzweigungsstrategie (Branching-Konzept), ebenso wie die automatisierte Versionierung (z. B. semantische Versionierung) von Build-Artefakten.

- **Automatisierung des Testprozesses**
  Im Bereich von Automatisierung ist auch die Automatisierung von Aktivitäten im Testprozess ein naheliegender Ansatz. Hierbei ist es wichtig, hervorzuheben, dass damit sämtliche Teststufen und Testarten gemeint sind: von der Qualitätssicherung von Anforderungen, Sicherstellung von ausreichender Überdeckung durch automatisierte Komponententests, statische Codeanalyse, automatisierte Validierung von Gesamtsystemen bis hin zu automatisierten Testaktivitäten im Produktivsystem (z.B. A/B-Tests). Auch die Erkennung von möglichen Sicherheitsrisiken und Angriffsvektoren kann durch Dynamic Application Security Testing (DAST), Static Application Security Testing (SAST), Dependency Scanning, Container Scanning oder Secret Detection sichergestellt werden.
- **Automatisierung der Infrastrukturprovisionierung**
  Infrastruktur als Code (IaC) mit den verbreiteten Werkzeugen wie Ansible, Chef, Terraform, Puppet, Kubernetes oder Ähnlichen gehört ebenso dazu wie Ansätze wie GitOps. Gerade hier ist erkennbar, wie eng der Betrieb mit anderen Teilbereichen des Softwarelebenszyklus zusammenarbeiten muss. Die Infrastruktur wird dadurch nämlich ein zentrales Entwicklungsartefakt und die Sicherstellung von dessen Qualität damit Aufgabe des Teams.
- **Automatisierung des Bereitstellungs- oder Verteilungsprozess**
  Dabei spielen nicht nur Prinzipien wie »Continuous Delivery« oder »Continuous Deployment« eine große Rolle, sondern es gilt auch andere Fragen zu klären. Dazu gehört zum Beispiel die Automatisierung eines Änderungsprotokolls (Changelog), Versionshinweise oder eine angemessene Archivierung von Artefakten zur Sicherstellung der Nachvollziehbarkeit.
- **Automatisierung des Überwachungsprozesses**
  Bereits zu Beginn der Entwicklung eines Produkts sollten Sie, wie erwähnt, darüber nachdenken, wie Sie es betreiben wollen, aber nicht nur das. Neben der Überwachung der Infrastruktur müssen Sie sich auch Gedanken über mögliche anwendungsspezifische Dinge machen. Die Analyse der Logdateien oder, noch spezifischer, das korrekte Schreiben von Logausgaben ist unerlässlich. Auch das Feedback von Kunden oder die Sammlung von Daten aus A/B- oder Funktionstests ist ein wichtiger Teil der Überwachung, der berücksichtigt werden sollte.

Lean

Entwicklungsteams verwenden Prinzipien aus Lean Development, um »Waste« zu eliminieren, indem sie den Mehrwert für Endbenutzer definieren und verstehen, wie er erreicht werden kann. Der Wertstrom wird optimiert, durch z.B. die Minimierung der parallel laufenden Arbeiten (WIP-Limit), die Sichtbarmachung und Nachvollziehbarkeit von Arbeit und Fortschritt, die Reduzierung der Komplexität bei der Übergabe und die Aufschlüsselung von Schritten, um sicherzustellen, dass der Strom der verbleibenden Schritte reibungslos, ohne Unterbrechungen und ohne Wartezeiten abläuft. Dazu gehört auch die Einführung funktionsübergreifender Teams und die Schulung von Mitarbeitern, die vielseitig und anpassungsfähig sind.

Measurement

Um die Möglichkeiten und etwaige Optimierungsmaßnahmen des gegenwärtigen Systems besser zu verstehen, sind gut durchdachte Metriken notwendig. Daten bzw. Informationen, die gesammelt und analysiert werden sollten, können Planungsdaten, Produktdaten, Qualitätsdaten oder allgemeinere Teamdaten sein. Grundprämisse ist dabei jedoch immer, dass diese Daten nicht einer Überwachung des Teams, sondern zu dessen kontinuierlicher Verbesserung dienen sollen. Dies ist wiederum nur dann möglich, wenn ausreichend Vertrauen und eine angemessene Fehlerkultur etabliert ist, andernfalls muss immer damit gerechnet werden, dass die Validität von Metriken nicht oder nur teilweise gegeben ist.

Im Sinne der kontinuierlichen Verbesserung sind somit folgende Aktivitäten hilfreich:

- Produkt- und systemspezifische Daten sammeln und analysieren
- Definieren von Metriken und Schwellenwerten
- Überwachung und Verfolgung von Metriken und Automatisierung von Benachrichtigungen
- Erkennen von Fehlern
- Quality Gates definieren und deren Einhaltung sicherstellen
- Eine Kultur des kontinuierlichen Lernens und der Verbesserung schaffen
- Verbesserung der Effizienz und Reduzierung der Zykluszeiten

Sharing

Klassischerweise gehört dazu die Etablierung einer Kultur ohne Schuldzuweisungen, was auf den ersten Blick einfach klingt, allerdings viel Erfahrung und Verständnis sowie gute Rollenbilder auf Managementebene erfordert.

Eine offene Kommunikationskultur sollte auch das Prinzip des Fragens und Teilens fördern. Gute (technische und organisatorische) Lösungen und Erfahrungen sollten im Team und zwischen den Teams geteilt werden, um die dadurch verbesserte Effizienz auch in weitere Teile der Organisation übertragen zu können.

Migrationsprojekte

Bei einem Migrationsprojekt steht eine Frage im Vordergrund: Funktioniert das System noch so wie zuvor? Die Beantwortung dieser Frage kann eine Testautomatisierung auf verschiedene Weise sinnvoll unterstützen.

Testdatengenerierung

Um Datenmigrationen frühzeitig testen zu können, müssen zwei Testdatensets vorbereitet werden:

- **Generierte synthetische Daten**
  Diese basieren auf den Migrationsregeln und testen somit strukturiert die implementierten Routinen des Datenimports. Sie sind Ergebnis der Testfallspezifikationsmethoden und können dann anhand der Werkzeuge generiert werden.
- **Produktionsdaten**
  Ein Test mit Produktionsdaten muss stattfinden, denn schlussendlich sollen diese Daten ja migriert werden. Es finden sich in Produktionsdaten immer wieder Konstellationen, die so in keinem Anforderungs- oder Entwurfsdokument spezifiziert wurden und bei einer Migration dann Probleme bereiten. Die Automatisierung kann hier zwei Aufgaben übernehmen: den Export der Produktivdaten aus dem alten System und, wenn nötig, eine Anonymisierung dieser Daten. Der Export kann später nach den Tests auch für die eigentliche Migration verwendet werden. Die Anonymisierung kann aus rechtlichen Gründen gefordert sein, da beispielsweise das Testteam gewisse Daten nicht sehen darf oder datenschutzrechtliche Anforderungen bestehen. Hier muss jedoch darauf geachtet werden, dass die Anonymisierung der Da-

ten den Aufbau der Daten beibehält und auch auf spezielle Eigenschaften wie Schreibweisen eingeht.

Die Ausbildung zum GTB Certified Test Data Specialist geht auf die komplexe Aufgabenstellung des Testdatenmanagements und insbesondere auf den Umgang mit schützenswerten Daten und Massendaten zu Testzwecken und zur Testautomation ein [Albrecht-Zölch 18; GTB 18].

Datenvergleich

Gerade wenn eine große Menge an Daten migriert wird, können die neuen Daten nicht manuell gegen die alten Daten geprüft werden. Der Vergleich muss daher automatisiert werden. Dies kann z.B. mit Komparatoren erfolgen, die auf die zu vergleichenden Daten auch gewisse Regeln anwenden können.

Ablaufvergleich

Die Einsetzbarkeit von Testautomatisierung zur Überprüfung der Funktionalität vor und nach einer durchgeführten Migration ist sehr stark von der Art der Migration abhängig. Wenn zum Beispiel dieselbe Applikation auf eine neue Systemplattform migriert wurde oder die Anbindung an eine andere Datenbank erfolgt ist, kann die Durchführung einer umfassenden, automatisierten Testsuite, die bereits für das bisherige System entwickelt wurde, in der neuen Umgebung sehr rasch die gewünschten Ergebnisse liefern. Ähnlich verhält es sich, wenn ein System z.B. automatisiert von Cobol nach Java transformiert wurde. Meist muss hier jedoch eine Anpassung der automatisierten Testfälle erfolgen und der Aufwand dafür ist abhängig davon, wie sehr die Entkopplung der fachlichen Testfälle von der technischen Implementierung gegeben ist.

Handelt es sich im Wesentlichen um einen Umstieg auf eine komplett neue Lösung, wird ein automatisierter Vergleich der Abläufe mit dem alten System unter Umständen aus Kosten-Nutzen-Sicht nicht zweckmäßig sein.

Migration der Testautomatisierung

Ebenso wie das System selbst kann auch eine vorhandene Testautomatisierung migriert werden. Wenn die Automatisierung schlüsselwortbasiert ist, genügt es, bei gleichbleibenden Abläufen die hinter den Schlüsselwörtern liegenden Skripte zu ändern. Eventuell müssen neue Schlüsselwörter erstellt oder alte Schlüsselwörter für obsolet erklärt werden, die Beschreibungssprache bleibt jedoch dieselbe. Somit müssen sich die Tester nicht umgewöhnen und können einen Teil der bereits automatisierten Testfälle weiternutzen.

Aus der Praxis:
Schrittweise Umstellung von Schlüsselwörtern

In einem agilen Altsystem-Ablöseprojekt wurden Testfälle schlüsselwortbasiert implementiert. Aufgrund der komplexen Workflows mussten teilweise Aktionen auf dem Altsystem durchgeführt werden, um automatisierte End-to-End-Testfälle umsetzen zu können. Im Zuge des Projekts wurde dieses Altsystem Schritt für Schritt abgelöst, und Schlüsselwort um Schlüsselwort konnte verändert werden, um anstelle des Altsystems nun das neu entwickelte Produkt anzusteuern. Aufgrund der ähnlichen Strukturen und Arbeitsabläufe der beiden Systeme konnte ein großer Teil der bestehenden Testfallabläufe beibehalten werden, auch wenn sich die durchzuführenden Aktivitäten und Technologien letztlich grundlegend unterschieden.