Andreas Spillner

Systematisches Testen von Software

Ein Überblick

Broschüre zum Jubiläum von »Basiswissen Softwaretest«

dpunkt.verlag

Prof. Dr.-Ing. Andreas Spillner (i. R.)
Andreas.Spillner@hs-bremen.de

Fellow der Gesellschaft für Informatik e. V.
Ehrenmitglied des German Testing Board e. V.
Präsidiumsmitglied im Arbeitskreis Software-Qualität und Fortbildung e. V.
Deutscher Preis für Software-Qualität 2022

4. Auflage 2023
Copy Editing: Ursula Zimpfer, Herrenberg
Satz und Herstellung: Frank Heidt
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck: www.wp-consult.eu, Mannheim

Artikel-Nr. 077.95725

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Andreas Spillner

Systematisches Testen von Software

Ein Überblick

4., aktualisierte Auflage

dpunkt.verlag

Grußwort

Als im September 2002 die erste Auflage des Buches von Andreas Spillner und Tilo Linz zum Thema »Softwaretesten« erschien, waren wir als Verlag eher skeptisch – erschien doch »Testen« damals nicht gerade beliebt zu sein, eher so etwas Lästiges wie Dokumentation von Software. Eine größere Nachfrage wurde demzufolge nicht erwartet.

Jetzt – 20 Jahre und fünf Auflagen später – wissen wir es besser: Nicht nur ist das Thema »Qualität von Software« und damit das Testen in den vergangenen Jahren immer wichtiger geworden, auch hat sich das Buch »Basiswissen Softwaretest« als das Standardwerk etabliert, das mittlerweile in der Printausgabe mehr als 75.000 Exemplare verkauft hat und inzwischen auch intensiv in der E-Book-Version eingesetzt wird.

Es hat unzähligen Softwareentwicklern und -entwicklerinnen dabei geholfen, das vom Buch abgedeckte ISTQB®-Zertifikat »Certified Tester – Foundation Level« zu erlangen, womit sich deren Chancen auf eine Karriere in der IT-Industrie deutlich verbessert haben dürften. Der Nutzen für die Fachleute in der Praxis zum Nachschlagen ist ebenso unbestritten wie seine Beliebtheit bei Informatikstudierenden der höheren Semester.

Wir bedanken uns bei den Autoren, dass sie damals den richtigen »Riecher« gehabt haben und seitdem das Buch über diesen langen Zeitraum gepflegt und weiterentwickelt haben. Wir beglückwünschen Andreas Spillner und Tilo Linz zum 20. Jubiläum ihres Buches und dass sie in diesem Jahr auch die Preisträger des Deutschen Preises für Software-Qualität (DPSQ) sind, der vom ASQF, GI-TAV und GTB jährlich vergeben wird. Das Thema bleibt weiterhin sehr wichtig und daher wünschen wir uns und der Leserschaft noch viele weitere Auflagen. Nicht zuletzt sei erwähnt, dass dieses Buch auch der Start unserer Buchreihe zum Certified-Tester-Curriculum geworden ist, die inzwischen über zehn Titel umfasst.

Ich bin sicher, das Buch und damit die Autoren haben dazu beigetragen, die (Software-)Welt ein kleines bisschen besser zu machen. Ihre Beliebtheit zeigt sich auch anhand dieser Broschüre, die nun auch schon die 4. Auflage erlebt.

Für das Team des dpunkt.verlags
Dr. Michael Barabas

Einleitung

In den zurückliegenden Jahren gab es im Bereich Qualitätssicherung und Testen viele neue Trends. Als Beispiele seien genannt: testgetriebene Entwicklung (Test-Driven Development), exploratives Testen und modellbasiertes Testen. Auch die agilen Vorgehensweisen haben dazu beigetragen, dass das Testen im Entwicklungsprozess an Bedeutung gewonnen hat. Entwicklung und Test werden im Team erarbeitet und durchgeführt. Eine strikte Aufgabentrennung ist nicht mehr vorhanden. Diese Trends bieten sowohl große Chancen als auch Herausforderungen in Test und Qualitätssicherung.

Das Testen von Software beansprucht 25–40 % des Entwicklungsaufwands, bei kritischen Systemen liegt der Anteil noch erheblich darüber. Leider ist aber in vielen Projekten festzustellen (belegt durch die Umfrage »Softwaretest in Praxis und Forschung« 2020), dass unabhängig vom Entwicklungsprozess der Einsatz von Testverfahren nicht immer in ausreichendem Umfang erfolgt. Zu oft werden die Testfälle noch unsystematisch und unstrukturiert erstellt. Der investierte Aufwand in das Testen wird dann nicht optimal genutzt.

Vielen Leserinnen und Lesern ist sicherlich das Aus- und Weiterbildungsschema zum »Certified Tester« bekannt, das in den letzten Jahren eine sehr große Verbreitung weltweit und in Deutschland erfahren hat und zu einem Quasistandard im Bereich des Testens geworden ist. Neben dem Kernprogramm mit ***Foundation, Advanced*** und ***Expert Level*** stehen aktuell Lehrpläne und damit Ausbildungsprofile für Spezialisierungen für das Testen in agilen Entwicklungen, im Automotive-Bereich sowie für Usability Test, Perfomance Testing, modellbasierten Test und Test mobiler Anwendungen zur Verfügung. Lehrpläne zum Testdaten-Spezialisten, Testautomatisierungsentwickler und Sicherheitstester runden das Angebot ab. Neben AI Tester und Acceptance Testing ist in 2022 neu der Software Development Engineer in Test (SDET) hinzugekommen, der sich speziell an Entwicklerinnen richtet. An dieser Entwicklung ist abzulesen, dass sich das Testen in den letzten Jahren weiter etabliert hat.

Im Bereich der Standardisierung existiert der »ISO/IEC/IEEE 29119 Software Testing«-Standard. Im Teil 4 »Test Techniques« sind insgesamt 16 Verfahren zur systematischen Herleitung von Testfällen definiert.

Sowohl der Lehrplan zum »Certified Tester – Foundation Level« als auch der Standard ISO 29119, Teil 4 dienen als Grundlage für die hier aufgeführten Testverfahren, die an Beispielen kurz vorgestellt werden.

Die Testverfahren sollen der Entwicklerin und dem Tester eine Hilfestellung geben, damit diese die für ihre Probleme passenden Tests in einem vertretbaren Zeitaufwand durchführen können, um die geforderte Qualität mit den Tests nachzuweisen. Ziel ist, angemessen statt aufwendig zu testen!

Entwicklerin und Tester stehen häufig vor folgenden Fragen:

- Welches Verfahren zum Entwurf der Tests oder welche Kombination von Verfahren soll ich verwenden?
- Wann kann ich das Testen als ausreichend ansehen und beenden?
- Welche Aufgaben muss ich erledigen, um das Testen strukturiert durchzuführen?

In der Broschüre werden diese Fragen beantwortet und aktuelle Trends im Testen vorgestellt. Allerdings kann nur ein erster Einstieg und ein grober Überblick gegeben werden. Entwicklerinnen und Tester finden erste Schritte hin zum systematischen Testen. Sie können so die von ihnen programmierten Softwarebausteine strukturiert prüfen und dadurch dazu beitragen, die Qualität der Software zu verbessern.

Hinweise zu weiterführender Literatur, zu Werkzeugen und Internetseiten finden sich am Ende der Broschüre.

Andreas Spillner
Bremen, im August 2022

Die relevanten Testmethoden verständlich und praxisnah erklären (Interview)

Zwanzig Jahre, sechs Auflagen, zehntausende Leserinnen und Leser: Diesen außerordentlichen Erfolg feiern Andreas Spillner und Tilo Linz mit ihrem Standardwerk »Basiswissen Softwaretest«. Anlässlich des Jubiläums haben wir mit den Autoren über das Buch und das Softwaretesten selbst gesprochen.

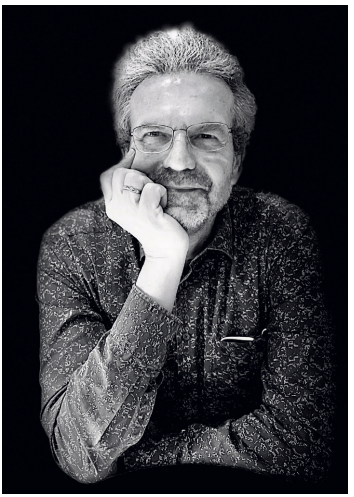
Herr Spillner, Herr Linz, Ihr Buch »Basiswissen Softwaretest« erschien am 18. September 2002 zum ersten Mal. Wir gratulieren zu diesem bemerkenswerten Jubiläum! In welcher Situation befand sich die Softwarebranche – und besonders der Bereich Softwarequalität und Testen –, als Sie beide sich für dieses Buch zusammengefunden haben?

Tilo Linz: Vor 20 Jahren war die Abhängigkeit von funktionierender Software noch nicht ganz so groß wie heute. Für medizintechnische Geräte wie Computertomographen, zugtechnische Systeme oder das Mobilfunknetz spielten Software und deren korrekte und zuverlässige Funktion schon damals eine entscheidende Rolle. Aber das Surfen im Internet über Smartphone oder Tablet gab es noch nicht. Dementsprechend spürte man im persönlichen Alltag diese Abhängigkeit von Software noch nicht in dem Ausmaß wie heutzutage.

Andreas Spillner: Damals wie heute sehen wir bei Softwarefirmen eine große Bandbreite bezüglich ihres Qualitätsbewusstseins und der Aufwendungen für Qualitätssicherung. Ein einheitliches Bild gab und gibt es daher nicht. Pauschal betrachtet ist insgesamt aber schon ein höheres Qualitätsbewusstsein in der Softwarebranche zu verzeichnen.

Was war schließlich der ausschlaggebende Punkt für Sie, das gemeinsame Buchprojekt zu initiieren?

Andreas Spillner: Meine Gründe waren folgende: Es war keine wirklich gute Literatur zu Qualitätssicherung oder Testen vorhanden. Das Standardwerk »The Art of Software Testing« von Glenford J. Myers (Erstauflage 1979) war schon sehr alt und präsentierte das Testen, wie es für die – aus heutiger Sicht – überschaubaren Softwareprogramme der 60er- und



Andreas Spillner war bis 2017 als Informatikprofessor in Bremen tätig und engagierte sich von Anfang an im German Testing Board. Bild: privat

70er-Jahre passend war. Und das Thema »Testen von Software« hatte bereits Fahrt aufgenommen: In der Gesellschaft für Informatik e.V. (GI) in Deutschland gab es schon die Fachgruppe für »Test, Analyse und Verifikation von Software« (TAV). Firmen und Fachverbände organisierten erste auf das Thema fokussierte Konferenzen. Zudem nahm die Industrie das immer bekannter werdende »Certified Tester«-Schema gut an.

Tilo Linz: Insbesondere Unternehmen aus den schon angesprochenen Branchen – wie Telekommunikation, Medizintechnik oder Bahntechnik – erkannten damals, dass eine gute Ausbildung der Mitarbeiter*innen im Fachgebiet Softwaretest ein Eckpfeiler ist, damit große Softwaresysteme überhaupt gebaut werden können. Dementsprechend häuften sich die Anfragen nach Schulungen, und es stieg der Bedarf, Testmethoden praxisgerecht zu trainieren und einzuüben. Da fehlte nur noch ein gutes Lehrbuch!

Andreas Spillner: Das »Certified Tester«-Schema bot auch den großen Vorteil, dass der Lehrplan für den »Certified Tester – Foundation Level« (CTFL) uns die Themen und Kapitel für das Buch vorgab. Sonst ist dies eine der schwierigen Entscheidungen als Autor: Was nehme ich ins Buch auf und was lasse ich weg?

Herr Spillner, Sie waren mehr als 40 Jahre im Bereich der Softwareentwicklung und -prüfung in Praxis und Forschung aktiv und haben an der Hochschule Bremen gelehrt. Herr Linz, Sie sind Vorstand und Mitgründer der imbus AG und seit mehr als 30 Jahren im Themengebiet Softwarequalitätssicherung tätig. Wie haben sich diese Einflüsse aus Wissenschaft und Praxis auf Ihr Buch ausgewirkt?

Tilo Linz darf in diesem Herbst nicht nur ein zwanzigjähriges Buchjubiläum, sondern auch den 30. Geburtstag seines Unternehmens imbus AG feiern.
Bild: imbus AG

Andreas Spillner: Wir haben – so meine Einschätzung – eine gute Mischung aus Theorie und Praxis im Buch erreicht. Wobei es nicht unbedingt die neuesten Forschungsergebnisse waren, die wir in die einzelnen Ausgaben jeweils aufgenommen haben.

Tilo Linz: Ja, es ging uns darum, die nach Stand der Technik relevanten Methoden möglichst verständlich und in praxistauglicher Form zu erklären.

Lange Zeit entwickelte ein Team die Software, während ein anderes Team davon weitestgehend losgelöst die Tests fuhr. Inwieweit verzahnen sich die Fachbereiche heute?

Tilo Linz: Im vergangenen Jahrtausend begann es zunächst so, dass es nur ein Team mit Programmierer*innen gab, die ihre Programme selbst testeten – mit entsprechend geringer Wirkung. Als Verbesserung der unbefriedigenden Situation wurden dann ab den 1990er-Jahren die Aufgabenbereiche und Teams zunehmend getrennt in »Design & Implementierung« und »Verifikation & Validierung«. Mit der Agilisierung der Softwareentwicklung ab den 2000er-Jahren verschmelzen die Aufgabenbereiche in den letzten Jahren nun wieder …

Andreas Spillner: … je nach Branche mit einem weiterhin spürbaren Übergewicht der Implementierungsseite. In der Beziehung hat sich somit nicht alles verändert. Es ist aber schon so, dass die »Gegensätze« zwischen Entwicklung und Test weiter abgebaut werden: Entwickler*innen müssen was vom Test verstehen und umgekehrt. Nur so kann aus einem Gegeneinander ein Miteinander werden, was unbedingt anzustreben ist.

Tilo Linz: Bei der heute verlangten Entwicklungsgeschwindigkeit ist das absolut notwendig und funktioniert in getrennten Silos auch nicht mehr.

Ihr Buch »Basiswissen Softwaretest« ist inzwischen in der 6. Auflage erhältlich und feiert in diesem Jahr sein 20. Jubiläum. Sie beide wurden kürzlich mit dem Deutschen Preis für Software-Qualität 2022 ausgezeichnet. Ein überwältigender Erfolg, zu dem wir herzlich gratulieren. Hand aufs Herz: Hatten Sie während der Arbeit an der ersten Auflage damit gerechnet, dass das Buch zu einem so erfolgreichen Standardwerk wird, das in der Praxis und gleichermaßen an Hochschulen eingesetzt wird?

Andreas Spillner: Nein, nie und nimmer.

Tilo Linz: Wir hatten auch nicht damit gerechnet, dass wir das Buch über 20 Jahre hinweg kontinuierlich aktualisieren würden, damit es über all die Zeit konform zum sich fortentwickelnden CTFL-Lehrplan ist.

Andreas Spillner: Auf einer Tagung zeigte mir ein Teilnehmer sein völlig zerfleddertes Buch voll mit handschriftlichen Notizen – ein wirkliches Arbeitsbuch, das täglich genutzt wurde. Ich kann mir kaum ein besseres Lob für ein Fachbuch vorstellen.

*Mehrere Auflagen und das ungebrochen große Interesse stehen auch für die hohe Relevanz des Softwaretestens. Während sicherheitskritische Anwendungen – etwa in Fahrzeugen oder beim Flugverkehr – glücklicherweise und auch zwingend notwendig sehr intensiv getestet werden, stehen wir aber beispielsweise bei mobilen Apps für Endanwender*innen oftmals vor kryptischen Bugs und Fehlermeldungen. Welche typischen Gründe nennen Ihnen Unternehmen dafür, dass das Testen vernachlässigt wurde? Was entgegnen Sie dann?*

Andreas Spillner: »Zu wenig Zeit«, »zu teuer, zu aufwendig«, »Testen kann Fehlerfreiheit ja nicht belegen« – dies sind einige der oft genannten Begründungen. Die Frage ist immer: »Wie viel Aufwand für das Testen ist gerechtfertigt?« Schon vor fast 30 Jahren formulierte die passende Antwort dazu der US-amerikanische Informatiker und Autor Jerry Weinberg: »Compared to what?« (Im Vergleich wozu?) Es kommt also immer darauf an, in welchem Umfeld die Software eingesetzt wird ...

Tilo Linz: … und welcher Schaden durch einen Bug eintreten kann. Wenn die App der Endanwender*in eine Online-Banking-App ist, dann ist auch hier ein professioneller Softwaretest notwendig. Das wird heute aber in aller Regel bei Apps für solche sensiblen Anwendungsfelder auch gemacht.

Herr Spillner, Sie haben die Studie »Softwaretest in Praxis und Forschung« maßgeblich mitgestaltet. Was sind die wichtigsten Entwicklungen der vergangenen 20 Jahre auf diesem Gebiet?

Andreas Spillner: Die Entwicklung verlief leider eher enttäuschend, es ging in kleinen Schritten über die Jahre voran. Bei der letzten Umfrage im Jahr 2020 stellten wir fest, dass der Einsatz und sogar die Kenntnis systematischer Testverfahren zurückgegangen ist. Sogar bei den Tester*innen, was besonders frustrierend ist.

Woran könnte dies liegen, Ihrer Einschätzung nach?

Andreas Spillner: Nur meine Vermutungen, die ich nicht belegen kann: Die Abkehr von der prozess- und phasenorientierten Vorgehensweise bei der Softwareentwicklung insgesamt hat sich sicherlich auch auf das

Testen ausgewirkt. Wenn die Erstellung des Programms im Team verantwortet wird und die eingesetzten Verfahren gemeinsam ausgewählt – oder weggelassen – werden, warum soll das Team dann beim Testen anders agieren? Ein weiterer Grund könnte sein, dass die automatisierte Durchführung der Testfälle eine gewisse Sicherheit gibt, dass ausreichend getestet wird. Nur: Wie die Testfälle erstellt wurden – ob zufällig oder systematisch – spielt eine eher untergeordnete Rolle.

Tilo Linz: Softwareentwicklung findet heute aber auch in viel größerem Umfang statt als früher, in unzähligen Anwendungsfeldern und in extrem schnellen Innovationszyklen. Wer da mithalten will, muss sich spezialisieren. Und nicht jeder spezialisiert sich auf Softwaretests.

Andreas Spillner: Bei den Kursen – wie übrigens auch im Buch – werden meist einfache Beispiele oder Übungen zu den einzelnen Verfahren durchgeführt. Dies geht in der Regel aus Zeitgründen auch nicht anders. Ich durfte über mehrere Jahre bei einem großen deutschen Softwarehaus als Referent an der jährlich durchgeführten Test-School teilnehmen. Diese ging über eine Woche und hatte einen eher kleineren theoretischen Teil zu den Testverfahren. Die meiste Zeit nutzten wir für die konkrete Umsetzung und Anwendung der Verfahren. Hierzu brachten die Teilnehmer*innen ihre aktuellen Projekte mit und wendeten die vorgestellten Verfahren an. Ein solches Vorgehen geht vermutlich am besten bei Inhouse-Schulungen und lediglich dann, wenn ausreichend Zeit für die Weiterbildung zur Verfügung steht. Nur wenn man neu gelernte Verfahren direkt anwendet, können diese in den Testalltag Einzug halten – andernfalls geraten sie schnell wieder in Vergessenheit.

In Ihren Büchern legen Sie Wert darauf zu zeigen, dass systematisches Testen, Testmanagement und Testen im agilen Umfeld spannend und kreativ sein kann. Herr Linz, wie haben Sie die Relevanz des Softwaretestens in den vergangenen 20 Jahren wahrgenommen?

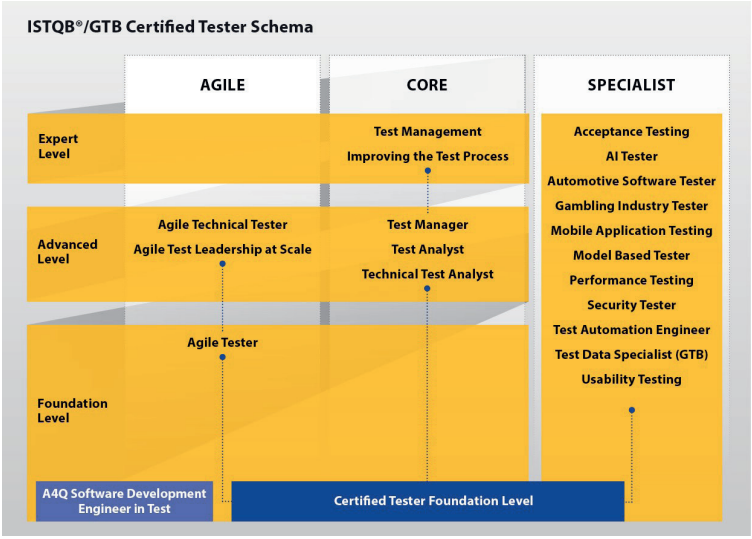
Tilo Linz: In den 90er-Jahren mussten wir tatsächlich den Nutzen von Test- und Qualitätssicherungsmaßnahmen gut verargumentieren und mit ROI-Berechnungen untermauern. Heute wissen die meisten Kunden, dass es ohne systematisches Testen nicht geht. Die Fragen heute sind: Wie kann ich meine Softwareentwicklung weiter beschleunigen, wie kann

ich meine Tests automatisieren, damit sie in einer vollautomatisierten DevOps-Pipline mit hoher Testabdeckung rund um die Uhr laufen.

Sie beide haben das »Certified Tester«-Programm des International Software Testing Qualifications Board (ISTQB®) im deutschsprachigen Raum maßgeblich etabliert. Welche Bedeutung hat es seit Gründung im November 2002 inzwischen in der IT erlangt?

Andreas Spillner: Die weit mehr als 100.000 abgelegten Prüfungen im ISTQB® Certified-Tester-Programm allein in Deutschland belegen die Bedeutung des Programms und die damit einhergehende Verbreitung des Fachwissens im Bereich Qualitätssicherung sehr klar.

Tilo Linz: Auch wenn Sie sich Stellenausschreibungen in diesem Themenfeld ansehen, dann stellen Sie fest, dass der »Certified Tester« ein gefordertes Must-have-Kriterium ist.



Certified Tester Schema (Bild: German Testing Board, german-testing-board.info)

Gibt es weitere Themen und Projekte, die Sie aktuell und in Zukunft beschäftigen werden?

Andreas Spillner: Als Pensionär genieße ich den Ruhestand und die Freiheit, mir die Themen und Projekte aussuchen zu können. Derzeit liegen keine größeren Projekte an. Mit einer Ausnahme: die 7. Auflage des Buches.

Tilo Linz: Die Prüfung und Absicherung von KI-basierten Systemen wird immer wichtiger. Hier engagieren wir uns bei imbus aktuell in zwei Forschungsprojekten. In einem der Projekte geht es um das Testen von KI-Algorithmen für Agrarmaschinen. Die Systeme sollen etwa lernen, wie stark Landwirte ihre Felder düngen müssen. Und die Forschungsfrage lautet: Wie prüfen wir, ob und was die Maschinen tatsächlich gelernt haben? Und ist das dann zielführend? Im zweiten Projekt steht die Frage »Wie kann man die Testfallexplosion beim Testen hochautomatisierter Fahrfunktionen in den Griff bekommen?« im Fokus. Hier arbeiten wir gemeinsam mit unseren Forschungspartnern an neuen Algorithmen, die helfen, aus der potenziell unendlichen Menge von Fahrszenarien die auszuwählen, die im Test die höchste Aussagekraft haben.

Herr Spillner, Herr Linz, wir danken Ihnen sehr für dieses Gespräch und wünschen Ihnen für alle künftigen Projekte viel Erfolg.

Das Interview führte der dpunkt.verlag.

Testen

Testen umfasst immer zwei Aspekte:

- **Qualitätsnachweis:** Es soll nachgewiesen werden, dass sich die Software konform zu den Anforderungen verhält.
- **Fehlerfindung:** Es sollen mögliche Fehler gefunden werden, bevor die Software eingesetzt wird.

Beides ist gleich wichtig. Fehler sollen so früh wie möglich erkannt und beseitigt werden, bevor sie Schaden beim Einsatz verursachen oder die Entwicklung verzögern und zusätzlichen Aufwand erfordern. Der Qualitätsnachweis stellt sicher, dass die Software den Anforderungen des Kunden entspricht und im Zusammenspiel mit den anderen Systemteilen »funktioniert«.

Mit Testen ist das Ausführen eines Programm(teil)s auf einem Rechner gemeint. In aller Regel testet jeder Entwickler sein von ihm programmiertes Softwareteil, bevor er es eincheckt und für die weitere Verwendung freigibt. Dieser Test ähnelt leider allzu oft einem unsystematischen Ausprobieren. Testfälle werden nicht auf Grundlage von Testverfahren entworfen, Negativtests werden nicht bedacht, erwartete Ergebnisse werden nicht vorab festgelegt, um nur einige der Nachteile eines solchen Vorgehens zu nennen. Eine fundierte Aussage über die durchgeführten Tests lässt sich so nicht geben.

Da beim Testen nicht alle Möglichkeiten der Anwendung der Software geprüft werden können – ein »Austesten« ist selbst bei kleineren Systemen nicht durchführbar –, müssen die Testfälle möglichst so gewählt werden, dass ein breites Spektrum des späteren Einsatzes vorab geprüft wird. Hier helfen Testverfahren bei der Herleitung der Testfälle und durch die Angabe von Kriterien bei der Beendigung des Testens.

Im Folgenden werden einige in der Praxis bewährte methodische Ansätze vorgestellt, um vom »Ausprobieren« zum »systematischen Testen« zu gelangen. Weitere Testverfahren sind in der entsprechenden Literatur ausführlich beschrieben. Hier kann nur ein Ausschnitt gezeigt werden, der »Appetit« auf mehr machen soll.

Testverfahren

Testverfahren dienen zum Entwurf von Testfällen, deshalb werden sie auch als Testentwurfsverfahren bezeichnet. Wir bleiben aber bei dem Begriff Testverfahren. Bei ihnen wird zwischen statischen und dynamischen Verfahren unterschieden. Statische Verfahren analysieren das Testobjekt, ohne es auf dem Rechner auszuführen. Manuelle Reviews und der Einsatz von statischen Analysewerkzeugen gehören zu diesen Verfahren.

Bei den dynamischen Verfahren wird das Testobjekt mit Testdaten versehen und auf dem Rechner ausgeführt. Beruht die Erstellung der Testfälle auf den Anforderungen bzw. der Spezifikation des Testobjekts, dann gehören diese Verfahren zu den anforderungsbasierten oder spezifikationsbasierten Testverfahren (Black-Box-Testverfahren). Dient der Programmtext – genauer die Programmstruktur – als Grundlage für die Überlegungen zur Testfallerstellung, dann wird von strukturbasierten Testverfahren oder White-Box-Testverfahren gesprochen.

Spezifikationsbasierte Testverfahren

Anhand von Beispielen werden im Folgenden einige spezifikationsbasierte Testverfahren kurz vorgestellt.

Äquivalenzklassen

Die Bildung von Äquivalenzklassen für die Ein- bzw. Ausgabeparameter eines Softwarebausteins gehört sicherlich zu den Verfahren, die jeder Entwickler auch rein intuitiv anwendet. Allerdings wird bei der systematischen Anwendung ein »Vergessen« der Werte, die außerhalb des Definitionsbereichs der Parameter liegen, verhindert.

Folgendes Vorgehen ist zur Herleitung der Testfälle gegeben: Für jeden Parameter ist der Definitionsbereich festzulegen. Er ist in weitere Unterbereiche aufzuteilen, von denen erwartet wird, dass sich das Testobjekt bei allen Werten des Unterbereichs gleich verhält oder, anders formuliert, sich äquivalent verhält. Für das Testergebnis ist es daher ohne Bedeutung, mit welchem Wert aus dem Unterbereich – der Äquivalenzklasse – der Test durchgeführt wird.

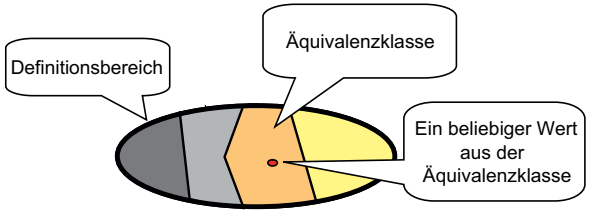


Abb. 1 Das orangefarbene Feld zeigt eine von vier Äquivalenzklassen, in die der Definitionsbereich zerlegt wurde.

Ein kleines Beispiel soll das Vorgehen verdeutlichen:

In einer Rabattaktion wird bei einem Warenwert von mehr als 25 Euro ein Rabatt von 15 % gewährt, bei einem Wert von über 150 Euro erhält der Käufer einen Rabatt von 25 %.

Zuerst sind die Äquivalenzklassen (ÄKx) zu bilden:

ÄK1: 0 <= Warenwert <= 25 (in Euro) ergibt keinen Rabatt.
ÄK2: 25 < Warenwert <= 150 ergibt einen Rabatt von 15 %.
ÄK3: 150 < Warenwert ergibt einen Rabatt von 25 %.

Aus der umgangssprachlichen Beschreibung wird die mathematische Formulierung der Wertebereiche abgeleitet, die präzise und überschneidungsfrei die Äquivalenzklassen des Definitionsbereichs darstellt. Nun können wir beliebige Repräsentanten aus den Äquivalenzklassen zur Definition von Testfällen (TFx) auswählen. Zunächst darf aber nicht vergessen werden, sich vorab Gedanken zum erwarteten Ergebnis zu machen! Denn nach der Durchführung der Testfälle muss ja verglichen werden, ob die tatsächlich berechneten Werte mit den erwarteten übereinstimmen oder ob ein Fehler vorliegt.

TF1: Eingabewert: 22,50 Euro; erwartetes Ergebnis: 22,50 Euro
TF2: Eingabewert: 30,00 Euro; erwartetes Ergebnis: 25,50 Euro
TF3: Eingabewert: 200,00 Euro; erwartetes Ergebnis: 150,00 Euro

So weit, so gut, und bis jetzt ist nichts wirklich Überraschendes geschehen. Jede oder jeder hätte wohl auf diese Testfälle ohne große Probleme kommen können. Aber wir sind noch nicht fertig. Die bisher aufgestellten

Testfälle behandeln nur die »Gut-Fälle«, d.h. die Eingabewerte, die innerhalb des Definitionsbereichs liegen. Leider kann nicht davon ausgegangen werden, dass nur solche gültigen Werte eingegeben bzw. zur Berechnung herangezogen werden. Um ein robustes Programm zu erhalten, müssen auch Fehleingaben behandelt werden.

Unsere Bildung der Äquivalenzklassen ist daher noch nicht abgeschlossen, bisher haben wir nur »gültige« Äquivalenzklassen erstellt. Was ist aber mit negativen Werten oder mit Werten, die den Zahlenbereich des Rechners (kleinste bzw. größte im Rechner darstellbare Zahl) unter- bzw. überschreiten? Wir müssen noch Klassen für ungültige Parameterwerte (uÄKx) ergänzen:

uÄK1: Min_Zahl <= Warenwert < 0 Ausgabe einer Fehlermeldung

Die gültige ÄK3 kann nun auch präzisiert und mit einer oberen Grenze versehen werden:

ÄK3: 150 < Warenwert <= Max_Zahl ergibt einen Rabatt von 25 %.

Weitere Äquivalenzklassen mit ungültigen Werten sind:

uÄK2: Warenwert < Min_Zahl
uÄK3: Warenwert > Max_Zahl,

wobei die beiden letzten Äquivalenzklassen sich je nach Beschaffenheit der Eingabeschnittstelle nicht immer in ausführbare Testfälle umformen lassen.

Auch ist es sinnvoll und erleichtert das Testen, wenn bereits in den Anforderungen gültige Ober- und Untergrenzen festgelegt werden, z.B. 15.000 Euro für die Obergrenze des Warenwerts:

ÄK3: 150 < Warenwert <= 15.000 ergibt einen Rabatt von 25 %.

Die ergänzende ungültige Äquivalenzklasse wäre dann:

uÄK4: 15.000 < Warenwert <= Max_Zahl Fehlermeldung.

Mit dem systematischen Vorgehen haben wir insgesamt sieben Äquivalenzklassen ermitteln können: drei gültige und vier ungültige Äquivalenzklassen. Somit wird unser Programm intensiver getestet.

Grenzwerte

Aber haben wir »spannende« Testdaten, die möglicherweise Fehler verursachen, verwendet? Sind nicht die Werte interessant, die ein »Umschalten« des Programmverhaltens bewirken? Genau, dieses Verfahren wird als Grenzwertanalyse bezeichnet. Es werden nicht beliebige Werte einer Äquivalenzklasse gewählt, sondern die Grenzwerte.

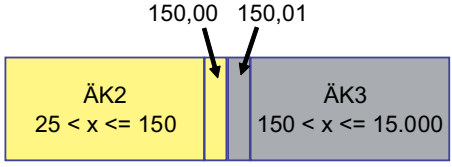


Abb. 2 Grenzwerte zwischen zwei Äquivalenzklassen (x = Warenwert)

Für die Grenze zwischen 15 % und 25 % Rabatt wollen wir uns die Testfälle überlegen. Dabei wird versucht, so dicht wie möglich an die Grenze oder genau auf die Grenze zu kommen.

TF1: Eingabewert: 150,00 Euro; erwartetes Ergebnis: 127,50 Euro (15 %)
TF2: Eingabewert: 150,01 Euro; erwartetes Ergebnis: 112,51 Euro (25 %)

Um auf der sicheren Seite zu sein, kann auch folgender Testfall ergänzend gewählt werden (drei Testfälle je Grenze):

TF3: Eingabewert: 149,99 Euro; erwartetes Ergebnis: 127,49 Euro (15 %)

Sind alle Äquivalenzklassen und alle Grenzen durch Testfälle überprüft, kann das Testen mit dem Verfahren als ausreichend angesehen werden. Allerdings ist zu bedenken, dass zum Beispiel die Kombination von Repräsentanten der Äquivalenzklassen unterschiedlicher Eingabeparameter nicht berücksichtigt wurde. Hierzu empfiehlt es sich, andere Verfahren anzuwenden.

Entscheidungstabellen

Beeinflussen Parameter sich untereinander oder sind Kombinationen von Bedingungen zu berücksichtigen, sind die Testfälle mittels Entscheidungstabellen abzuleiten.

Am besten wir machen uns das Vorgehen wieder an einem Beispiel klar: Um Kunden an ein Geschäft zu binden, werden Kundenkarten an Stammkunden ausgegeben, deren Inhaber einen Rabatt von 7 % auf alle Waren erhalten. Um den Verkauf direkt nach Öffnung (8 Uhr) anzukurbeln, wird in den ersten beiden Verkaufsstunden ein Rabatt von 5 % gewährt. Beide Rabatte können kombiniert werden. Alle Kunden, die vor 10 Uhr einkaufen, erhalten zusätzlich ein kleines Werbegeschenk.

Bedingungen				
Stammkunde	JA	JA	NEIN	NEIN
Zeit zwischen 8 und 10 Uhr	JA	NEIN	JA	NEIN
Aktionen				
Rabatt von 7%		X		
Rabatt von 5%			X	
Rabatt von 12% (5% + 7%)	X			
Werbegeschenk	X		X	

Tab. 1 Entscheidungstabelle

Es sind zwei Bedingungen gegeben, die kombiniert werden müssen. Eine Entscheidungstabelle ist anzufertigen, die aus vier Bereichen besteht: Oben sind die Bedingungen und alle Kombinationsmöglichkeiten (Ja/Nein) aufgeführt; im unteren Teil der Tabelle finden sich die Aktionen bzw. erwarteten Ausgaben. Ein »x« zeigt an, bei welchen Bedingungskombinationen welche Aktionen zutreffen.

Bei zwei Bedingungen ergeben sich insgesamt vier Kombinationen. Da im Beispiel bei jeder Kombination eine andere Zusammenstellung von Aktionen auszuführen ist, kann auf keine Kombination verzichtet werden.

Jede Spalte der rechten Seite der Entscheidungstabelle entspricht einem Testfall, und an den Kreuzen im unteren Teil lassen sich auch die erwarteten Ergebnisse bzw. Aktionen direkt ablesen.

Sind mehrere Bedingungen und deren Kombinationen zu berücksichtigen, nimmt die Entscheidungstabelle an Umfang zu. Allerdings lassen sich dann oft Kombinationen streichen, weil sie zum Beispiel irrelevant sind oder keinen Einfluss auf die Ergebnisse haben.

Nehmen wir an, in unserem Beispiel handelt es sich um einen Spirituosenladen, der nur an Personen ab 18 Jahren Ware verkaufen kann. Die zusätzliche Bedingung (18 Jahre oder älter) ist aufzunehmen. Allerdings führt dies nicht zu einer Verdoppelung der Testfälle, da bei Nichterfüllung der Altersgrenze alle anderen Bedingungen obsolet sind.

Bedingungen					
Alter >= 18	NEIN	JA	JA	JA	JA
Stammkunde		JA	JA	NEIN	NEIN
Zeit zwischen 8 und 10 Uhr		JA	NEIN	JA	NEIN
Aktionen					
Kein Warenverkauf	X				
Rabatt von 7%			X		
Rabatt von 5%				X	
Rabatt von 12% (5% + 7%)		X			
Werbegeschenk		X		X	

Tab. 2 Entscheidungstabelle mit weiterer Bedingung

Mithilfe von Entscheidungstabellen lassen sich Vollständigkeit und Redundanzfreiheit der Tests überprüfen. Sie bieten daher ein sehr einfaches, aber formales Mittel zur Spezifikation von Testfällen, die direkt aus der Tabelle abgelesen werden können. Sind alle Spalten der rechten Seite der Tabelle für die Erstellung eines Testfalls verwendet, ist der Testaufwand als ausreichend anzusehen, da alle relevanten Bedingungen und deren Kombinationen getestet wurden.

Klassifikationsbaumverfahren

Wenn alle Parameter unabhängig voneinander sind, also frei kombinierbar, dann ist es schwierig, die Übersicht zu behalten. Mithilfe des Klassifikationsbaumverfahrens gelingt dies.

Die Grundidee für das Verfahren ist die Zerlegung des Eingabedatenraums des Testobjekts in Klassifikationen und Klassen. Klassifikationen sind für den Test als relevant erachtete Gesichtspunkte, die weiter in Klassen zerlegt werden. Die Klassen entsprechen im Beispiel den oben beschriebenen Äquivalenzklassen. Darauf aufbauend werden durch Kombination der Klassen unterschiedlicher Klassifikationen Testfälle definiert.

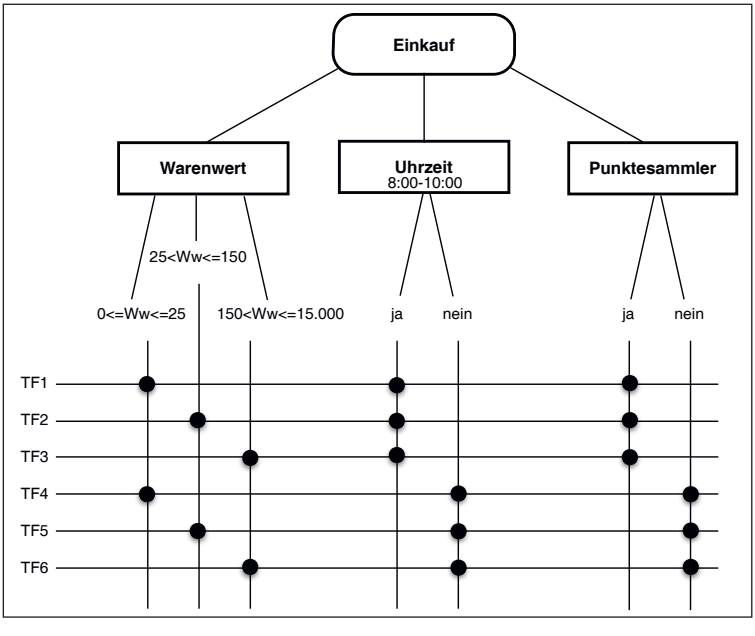


Abb. 3 Klassifikationsbaum für einen Einkauf

Bleiben wir bei unserem Beispiel mit der Rabattierung in Abhängigkeit vom Warenwert und der Einkaufszeit, beschränken uns aber auf gültige Werte. Nehmen wir an, dass noch ein Punktesammelsystem zu berücksichtigen ist, d.h., dass das Programm bei einem Punktesammler seine durch den Kauf erworbenen Punkte im System verarbeiten muss. Es ergeben sich also drei Klassifikationen mit drei bzw. zwei Klassen.

Abbildung 3 zeigt den Klassifikationsbaum mit sechs Testfällen (TF1-TF6). Durch die grafische Darstellung ist sofort zu sehen, welche Werte der drei Klassifikationen (in dem Fall Parameter) miteinander zu einem Testfall kombiniert werden. Wie ebenfalls zu erkennen ist, gehen morgens (8–10 Uhr) nur Punktesammler und in der restlichen Verkaufszeit keine Punktesammler einkaufen. Hier fehlen also noch weitere Testfälle!

Die grafische Darstellung hilft, bei der Vielzahl von Kombinationsmöglichkeiten noch die Übersicht zu behalten. Bei jedem Testfall ist ersichtlich, aus welchen Klassen er zusammengesetzt ist. Ein einfacher, schneller Überblick ist dadurch gegeben. Selbstverständlich sind zu jedem Testfall noch die erwarteten Ergebnisse vorab festzulegen.

Der Klassifikationsbaum ist unter Verwendung eines Werkzeugs (z.B. dem freiverfügbaren *TESTONA Light*) zu erstellen. Das Werkzeug kombiniert automatisch aus den Klassen Testfälle. Dabei können auch Abhängigkeiten zwischen den Elementen des Klassifikationsbaums durch aussagenlogische Formeln berücksichtigt werden. Darüber hinaus können die Klassen des Klassifikationsbaums mit Auftretenswahrscheinlichkeiten, Fehlerwahrscheinlichkeiten oder Risikowerten annotiert werden, die bei der Testgenerierung entsprechend berücksichtigt werden.

Kombinatorisches Testen

Die Basis der bisherigen Überlegungen zur Kombination von Parametern waren die Spezifikation und die darauf aufbauenden Überlegungen. Beim kombinatorischen Testen wird die Mathematik zu Hilfe genommen, um eine Auswahl von Kombinationen zu treffen.

Vereinfachen wir unser Beispiel und berücksichtigen nur einen Warenwert ab 150 Euro für die Rabattierung, dann ergeben sich für die drei Parameter insgesamt acht mögliche Kombinationen (s. Tab. 3).

Kombination / Testfall	Warenwert > 150 Euro	Frühkauf 8.00-10:00	Punkte-sammler
1	JA	JA	NEIN
2	JA	JA	JA
3	JA	NEIN	NEIN
4	JA	NEIN	JA
5	NEIN	JA	NEIN
6	NEIN	JA	JA
7	NEIN	NEIN	NEIN
8	NEIN	NEIN	JA

Tab. 3 Alle Kombinationen für drei Parameter

Sehen wir uns die folgende Tabelle 4 mit den vier Kombinationen näher an. Fällt Ihnen was auf?

Kombination / Testfall	Warenwert > 150 Euro	Frühkauf 8.00-10:00	Punkte-sammler
1	JA	JA	NEIN
4	JA	NEIN	JA
6	NEIN	JA	JA
7	NEIN	NEIN	NEIN

Tab. 4 Vier Kombinationen für drei Parameter, paarweiser Test

Alle vier möglichen Kombinationen zwischen dem Warenwert und dem Frühkauf, zwischen dem Warenwert und dem Punktesammler und auch zwischen dem Frühkauf und dem Punktesammler kommen in den vier Testfällen vor.

Die Kernidee des kombinatorischen Testens ist, nicht alle möglichen Kombinationen über alle Parameter beim Testen zu berücksichtigen, sondern nur Kombinationen zwischen zwei, drei oder mehreren Parametern, diese dann aber vollständig.

Im Beispiel wurden jeweils zwei Parameter vollständig miteinander kombiniert, dieses Vorgehen wird als »paarweises Testen« bezeichnet. Ziel des paarweisen Testens ist die Entdeckung aller Fehler, die auf der Interaktion zweier Parameter beruhen. Die Einsparung ist hier noch relativ gering mit 50 % weniger Testfälle im Vergleich zur vollständigen Kombination. Im Buch »Basiswissen Softwaretest« gibt es ein ausführliches Beispiel, bei dem 150.000 mögliche Kombinationen auf 100 Testfälle durch das paarweise Testen reduziert werden.

Weitere kombinatorische Verfahren berücksichtigen 3er- oder mehr Kombinationen, um die Zahl der Testfälle einzuschränken, aber trotzdem eine gewisse Testbreite zu garantieren. Im erwähnten Beispiel sind 1.014 Testfälle erforderlich, um alle 3er-Kombinationen abzudecken. Das frei verfügbare Werkzeug *Advanced Combinatorial Testing System (ACTS)* des amerikanischen National Institute of Standards and Technology liefert die entsprechenden Kombinationen.

Zustandsübergangstest

Neben Parameterwerten und deren Kombinationen sind häufig auch Zustände (von Objekten) für den Programmablauf und damit auch für den Test zu berücksichtigen. Auch dieses Vorgehen soll an einem Beispiel aus der Geschäftswelt erörtert werden.

Basis für die Testüberlegungen ist das Zustandsdiagramm, das Zustände und deren Übergänge grafisch darstellt (s. Abb. 4).

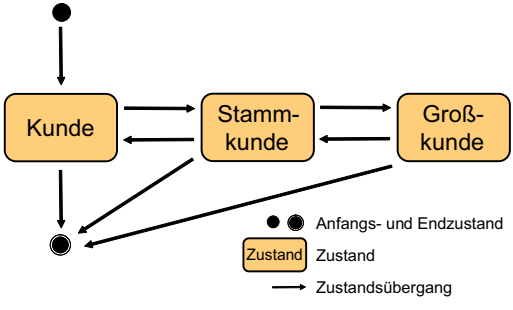


Abb. 4 Zustandsdiagramm

Bei einer Firma wird zwischen Kunden, Stammkunden und Großkunden unterschieden. Für jede Gruppe gibt es unterschiedliche Rabatt- und Zahlungsmodalitäten. Ein Neukunde gehört zur Gruppe der Kunden, bei regelmäßigen Einkäufen wird er zum Stammkunden, und übertrifft seine Kaufsumme in einem festgesetzten Zeitraum eine bestimmte Höhe, wird er zum Großkunden. Geht seine Umsatzhöhe bzw. seine Kauffrequenz zurück, kann er wieder zum Stammkunden bzw. »einfachen« Kunden werden. Auch ist es möglich, Kunden, Stamm- und Großkunden zu verlieren, d.h. in den Endzustand des Zustandsdiagramms zu gelangen. Nicht möglich soll es sein, dass ein Kunde direkt zum Großkunden »aufsteigt« oder vom Großkunden direkt zum Kunden »degradiert« wird. Der Weg führt immer über den Stammkunden.

Wie ist nun so eine Gegebenheit zu testen? Grundlage ist das Zustandsdiagramm, das ein gerichteter Graph ist. Dadurch lassen sich Graph-Überdeckungsmaße anwenden. Eine Forderung ist, dass jeder Zustand (Knoten im Graphen) während des Testens erreicht wird oder dass jeder Zustandsübergang (Kanten im Graphen) zur Ausführung gebracht wird.

Um aus dem Diagramm Testfälle abzuleiten, ist ein Übergangsbaum zu erstellen, dessen Wurzel der Anfangszustand ist. Das Diagramm wird »abgelaufen«, und dabei werden die Zustände und Zustandsübergänge aufgesammelt, bis der Endzustand erreicht wird oder ein Zustand, der bereits vorkam.

Daraus ergeben sich folgende Abfolgen (Reihenfolge der Zustände):

A1: Anfangszustand, Kunde, Endzustand
A2: Anfangszustand, Kunde, Stammkunde, Kunde
A3: Anfangszustand, Kunde, Stammkunde, Endzustand
A4: Anfangszustand, Kunde, Stammkunde, Großkunde, Stammkunde
A5: Anfangszustand, Kunde, Stammkunde, Großkunde, Endzustand

Zwei weitere Abfolgen, die nicht im Graphen verzeichnet sind, aber beim Testen berücksichtigt werden müssen, sind:

A6: Anfangszustand, Kunde, Großkunde
A7: Anfangszustand, Kunde, Stammkunde, Großkunde, Kunde

Aus den Abfolgen sind dann entsprechende Testfälle abzuleiten, d.h., es müssen vom Anfangszustand ausgehend die entsprechenden Aktionen (Umsätze des Kunden) ausgeführt werden, sodass die in der Abfolge aufgeführten Zustände erreicht werden. Bei der Ausführung der Testfälle ist nicht nur der erreichte letzte Zustand der Abfolge zu prüfen, sondern auch die Zwischenzustände. Nur so kann nachgewiesen werden, dass die Abfolge wie vorgesehen auch zur Ausführung durch den Testfall gebracht wurde.

Die Abfolgen A6 und A7 müssen zu einer Fehlermeldung führen, da deren Reihenfolge nicht vorgesehen ist. Aus dem Zustandsdiagramm lässt sich auch eine Zustandsübergangstabelle ableiten, die dann zur Erstellung der Testfälle herangezogen werden kann.

Was fehlt?

Im Standard ISO 29119 bzw. im »Certified Tester – Foundation Level« werden darüber hinaus folgende Black-Box-Testverfahren aufgeführt:

- **Syntaxtest** prüft, ob die durch die Syntax vorgegebenen gültigen Eingaben von dem Testobjekt akzeptiert und ungültige Eingaben abgewiesen werden.
- **Ursache-Wirkungs-Graph-Analyse** basiert auf der Erstellung eines Graphen, der die komplexen logischen Zusammenhänge der Eingaben sichtbar macht und zur Testfallerstellung nutzt.
- **Szenariotest** basiert auf bekannten Abläufen des Testobjekts und setzt diese in Testfälle um.
- **Anwendungsfallbasierter Test**, bei dem aus Anwendungsfällen (***use cases***) Testfälle abgeleitet werden.
- **Zufallstest**, bei dem zufällige Testeingabedaten erzeugt werden und somit auch Fehler entdeckt werden können, auf die die Testerin sonst nicht gekommen wäre.

Strukturbasierte Testverfahren

Alle bisher vorgestellten Testverfahren basieren auf der Spezifikation bzw. den Anforderungen an das Testobjekt. Verfahren, die den Programmtext in die Testüberlegungen mit einbeziehen, sollen im Folgenden kurz vorgestellt werden.

Grundidee der Verfahren ist, dass über Programmteile, die beim Testen nicht zur Ausführung gekommen sind, keine Aussage bezüglich der Korrektheit getroffen werden kann. So fordert der Anweisungstest, dass jede Anweisung des Testobjekts mindestens einmal zur Ausführung gebracht wird. Beim Entscheidungstest müssen die Testfälle sicherstellen, dass jede Entscheidung (z. B. in einer IF-Anweisung) mindestens einmal mit dem Ergebnis *true* und mindestens einmal mit *false* zur Ausführung gekommen ist.

Die strukturbasierten Testverfahren sind sehr einsichtig, meist leicht zu verstehen, und es gibt eine große Anzahl von unterstützenden Werkzeugen zur Berechnung der erreichten Überdeckungsgrade. Auch lassen sich sehr einfach Kriterien zur Beendigung des Testens aufstellen, z. B. muss 85 % Anweisungsüberdeckung nachgewiesen werden.

Leider wird diesen Verfahren in der Praxis eine oft zu große Bedeutung beigemessen. Die geforderte Überdeckung bedeutet keinesfalls, dass alle Fehler in dem zu 100 % ausgeführten Programmstück aufgedeckt wurden.

Folgendes Vorgehen ist anzuraten: Es sollen zuerst die spezifikationsbasierten Verfahren zur Anwendung kommen. Parallel zur Ausführung dieser Testfälle ist die erreichte Codeüberdeckung mit einem Werkzeug zu messen. Danach sind ggf. ergänzende Testfälle zu erstellen, und zwar für die Programmteile, die bisher noch nicht zur Ausführung gekommen sind.

Zu den strukturbasierten Testverfahren gehören:

- **Anweisungstest**: Wenn alle Tests durchgeführt sind, dann soll jede Anweisung im Testobjekt mindestens einmal zur Ausführung gekommen sein, dann ist ein Überdeckungsgrad von 100 % erreicht.
- **Entscheidungstest**: Er verlangt, dass jede Entscheidung im Testobjekt nach Durchführung aller Tests mindestens einmal die Wahrheitswerte ***true*** und ***false*** angenommen hat, d. h., beide Entschei-

dungsausgänge sind beim Test zum Tragen gekommen. Hinweis: Eine Entscheidung einer IF-Anweisung hat immer zwei Entscheidungsausgänge, auch wenn der ELSE-Teil im Programmcode leer ist.

- Besteht eine Entscheidung aus mehreren (mindestens zwei) Bedingungen, die über boolesche Operatoren (AND, OR, ….) verknüpft sind, dann sind diese Entscheidungen mit folgenden Testverfahren separat zu überprüfen:
  - Der **Bedingungstest** verlangt, dass jede einzelne Bedingung (z. B. a>0) einer Entscheidung jeweils beide Wahrheitswerte (***true*** und ***false***) während der Durchführung aller Tests mindestens einmal angenommen hat.
  - Beim **Mehrfachbedingungstest** sind alle möglichen Kombinationen aller Bedingungen zu prüfen, d. h., besteht eine Entscheidung aus zwei Bedingungen, dann sind mindestens vier Kombinationen gegeben, die durch Testfälle abzudecken sind, bei drei Bedingungen sind es acht usw. Sind Abhängigkeiten zwischen den Bedingungen enthalten, so kann es vorkommen, dass nicht alle Kombinationen realisierbar sind.
  - Der **modifizierte Bedingungs-/Entscheidungstest** beschränkt sich bei der Kombinatorik der Bedingungen auf die Testfälle, bei denen eine Änderung einer Bedingung (z. B. von ***true*** auf ***false***) ebenfalls einen Änderung des Wahrheitswertes der gesamten Entscheidung zur Folge hat.

Im Standard ISO 29119 werden noch der **Zweigtest** und der **Datenflusstest** bei den strukturbasierten Testverfahren aufgeführt. Der Zweigtest ist analog zum Entscheidungstest zu sehen, nur dass der Kontrollfluss als Grundlage herangezogen wird. Der Datenflusstest zielt darauf ab, dass zwischen dem Entstehen eines Wertes und dessen Verwendung ein Datenfluss besteht. Dieser Fluss dient als Grundlage für die Erstellung der Testfälle. Unterschiedliche Intensitäten der Datenflussverfahren sind definiert.

Intuitives und erfahrungsbasiertes Testen

Neben den systematischen Verfahren zur Ermittlung der Testfälle sind Intuition und Erfahrung der Entwicklerinnen und Tester eine gute Quelle für zusätzliche Testfälle. Meist lassen sich so noch weitere Fehler nachweisen, die mit den methodischen Ansätzen nicht immer aufgedeckt werden.

Sind Berechnungen vorzunehmen, dann sind ein oder mehrere Testfälle mit null oder mit Werten durchzuführen, die sehr klein oder sehr groß sind, oder auch mit deren Kombination. Beim Testen von Sortierungen sind bereits sortierte Listen, umgekehrt sortierte Listen und Listen mit gleichen Elementen aussichtsreiche Kandidaten, um Fehler aufzudecken.

Die ISO 29119 führt die **Fehlererwartungsmethode** als Verfahren auf, bei der auf die bereits gemachten Erfahrungen zurückgegriffen wird.

Kriterien, wann eine ausreichende Zahl von Testfällen intuitiv erstellt worden ist, lassen sich im Gegensatz zu den systematischen Verfahren nicht angeben.

Risikobasiertes Testen

Dies ist kein Testverfahren im engeren Sinne zur Spezifikation von Testfällen, sondern eine systematische Auswahl der Testobjekte (Komponenten und/oder Funktionalität) und der Intensität des Testens. Diese erfolgt in Abhängigkeit des im Fehlerfall vermuteten Risikos. Je höher das vermutete Risiko bei nicht korrekten Arbeiten des Systems ist, desto intensiver (Einsatz von mehreren Testverfahren mit jeweils hoher Intensität) muss das entsprechende Systemteil getestet werden. Ein Verteilen der zeitlich und von den Ressourcen her oft beschränkten Testaktivitäten nach dem »Gießkannenprinzip« ist nicht wirklich sinnvoll, da dann kritische Systemteile zu wenig und unkritische zu intensiv getestet werden.

Untersuchungen haben ergeben, dass Fehler oft »gehäuft« vorkommen. Das heißt, wenn in einem Systemteil viele Fehler beim Testen nachgewiesen werden, dann lohnt es sich, noch weitere Tests durchzuführen. Die Wahrscheinlichkeit, zusätzliche Fehler aufzudecken, ist hoch. Der verwendete Testprozess muss solchen Gegebenheiten Rechnung tragen, und die Tester müssen flexibel reagieren können.

Testprozesse

Nachdem wir nun einige Verfahren zur Ermittlung der Testfälle kennengelernt haben, stellt sich die Frage, welche weiteren Aufgaben zum Testen noch gehören. In diesem Zusammenhang wird oft vom Testprozess gesprochen, der parallel zum »eigentlichen« Entwicklungsprozess durchgeführt wird. Für den Testprozess gibt es unterschiedliche Ansätze, die im Folgenden grob skizziert werden. Beim agilen Vorgehen sind die Aufgaben ebenfalls durchzuführen, wenn auch nicht immer ein Prozess im engeren Sinne verfolgt wird.

ISTQB®-Testprozess

Ein weltweit verbreiteter Testprozess ist vom ISTQB® (International Software Testing Qualifications Board) entwickelt worden. Er umfasst folgende Aktivitäten (s. Abb. 5):

- **Testplanung:** Es ist festzulegen, welche Teile des Systems wie intensiv und mit welchen Testverfahren getestet werden sollen. Die verfolgte Teststrategie wird somit bestimmt. Die benötigten Ressourcen (qualifiziertes Personal, Zeit, Werkzeuge, ...) sind ebenso einzuplanen.
- **Testüberwachung und Teststeuerung:** Während des gesamten Testprozesses wird die Einhaltung der Planung überwacht und es wird ggf. durch Korrekturen steuernd eingegriffen.
- **Testanalyse:** Es ist festzulegen, »was« zu testen ist. Dazu werden alle Dokumente, aus denen Anforderungen an das Testobjekt hervorgehen (Testbasis) untersucht, um zu erfüllende Kriterien an den Test zu identifizieren.
- **Testentwurf:** Festgelegt wird, »wie« zu testen ist. Erste Testfälle werden unter Nutzung von Testverfahren entworfen, ebenso wird die notwendige Testumgebung zur Durchführung der Tests aufgebaut bzw. zur Verfügung gestellt.
- **Testrealisierung:** Die Testfälle werden konkretisiert, d.h. mit den tatsächlichen Eingabewerten und den erwarteten Ausgaben bzw. dem Ausgabeverhalten spezifiziert. Testfälle werden zu Testsuiten gruppiert, um die Tests effizient ausführen zu können.

- **Testdurchführung:** Die Testfälle/Testsuiten werden manuell bzw. teil- oder vollautomatisiert auf dem Rechner zur Ausführung gebracht. Der »eigentliche« Test wird durchgeführt. Die Tests sind zu protokollieren und auszuwerten (Vergleich Soll- und Istergebnis, Messung der erreichten Überdeckungen, …). Da oft mehrere Hundert Testfälle durchgeführt werden, ist ein zusammenfassender Bericht hilfreich.
- **Testabschluss:** Die Archivierung der Testfälle, der Testdaten, der Testumgebung, der Testinfrastruktur und anderer Testmittel für eine spätere Wiederverwendung ist vorzunehmen. Eine kritische Auswertung des Testvorgehens ist nützlich, um Verbesserungspotenzial zu erkennen und künftig umzusetzen (»Lessons Learned«).

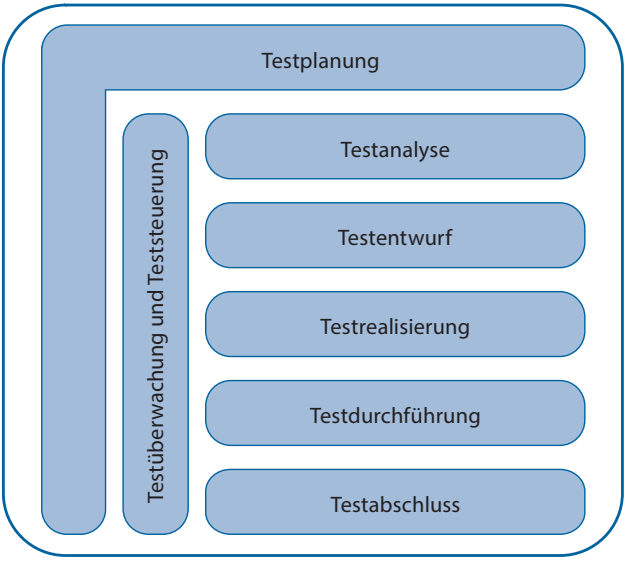


Abb. 5 ISTQB®-Testprozess

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

Testprozess nach ISO 29119

In der ISO 29119, Teil 2 ist der Testprozess nach einem hierarchischen Modell aufgebaut und in drei Bereiche unterteilt: Prozesse auf Organisationsebene, Testmanagementprozesse und Prozesse für den dynamischen Test (s. Abb. 6).

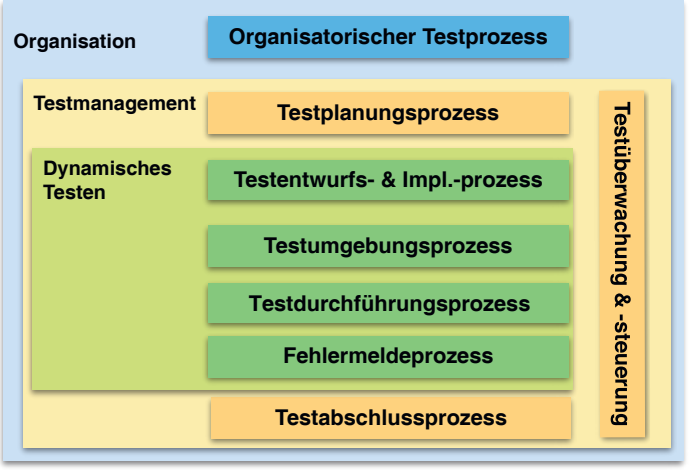


Abb. 6 Hierarchische Testprozesse der ISO 29119-2

Zu jedem Prozess sind die Eingangsdokumente, das Ziel, die Ergebnisse, die Aktivitäten sowie die Aufgaben definiert. Zum Beispiel gehören zum Testplanungsprozess insgesamt neun Aktivitäten, die der Testmanagerin obliegen.

Auf eine detaillierte Darstellung der Prozesse wird hier verzichtet und auf die entsprechende Literatur bzw. auf den Standard verwiesen.

Teststufen

Bei allen Projekten sind die Testaktivitäten auf unterschiedlichen Stufen (mit entsprechendem Abstraktionsniveau) durchzuführen. Dabei spielt es keine große Rolle, nach welchem Entwicklungsmodell die Software erstellt und getestet wird. In der Regel werden folgende Teststufen unterschieden:

- **Komponententest:** Ein einzelner Softwarebaustein (eine Komponente oder eine Gruppe von stark zusammenhängenden Komponenten) wird getestet; meist durchgeführt durch den Entwickler, weshalb diese Teststufe auch als Entwicklertest bezeichnet wird.
- **Integrationstest:** Die Prüfung der Schnittstellen zwischen den Komponenten(gruppen) steht im Mittelpunkt der Testaktivitäten. Die Komponenten werden schrittweise integriert und dabei ihr »Zusammenspiel« getestet.
- **Systemtest:** Ist die Integration abgeschlossen, wird das System als Ganzes getestet. Es wird entschieden, ob das System bereit zur Abnahme (BzA) ist oder ob noch Korrekturen vorzunehmen sind, da gravierende Fehler auf Systemebene auftreten.
- **Abnahmetest:** Die Abnahme des Systems erfolgt durch den Kunden auf Grundlage der Anforderungen. Auf dieser Teststufe steht mehr die (Nutzungs-)Qualität des Systems als der Nachweis von Fehlern im Fokus.

Für jede Stufe ist der Testprozess anzuwenden, allerdings mit unterschiedlichen Zielsetzungen, Testverfahren sowie Dokumenten, gegen die getestet wird. Die Teststufen werden häufig als sogenannte Testpyramide (s. Abb. 7) dargestellt, um die Anzahl und den Aufwand bei der Erstellung der Testfälle, den Grad der Automatisierung und die Laufzeit pro Testfall auf der jeweiligen Stufe zu verdeutlichen.

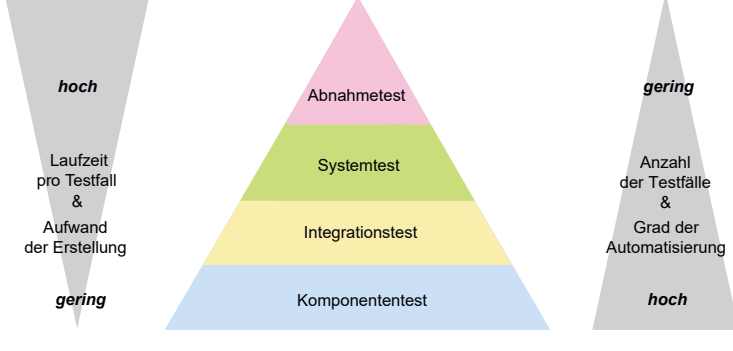


Abb. 7 Testpyramide

Was ist mit …

… einer ganzen Reihe von Testverfahren und anderen aktuellen Themen aus dem Bereich Test, die bisher noch nicht erörtert wurden? Dies soll hier – allerdings nur recht kurz – für einige Themen nachgeholt werden.

… modellbasiertem Test?

In den letzten Jahren kommen vermehrt Modelle (meist UML-Modelle) beim Entwicklungsprozess zum Einsatz. Use-Case-, Aktivitäts- oder Sequenzdiagramme, um nur einige zu nennen, dienen zur Klärung und Verdeutlichung des Sachverhalts. Mithilfe der Modelle lassen sich bereits vor der Programmierung Inkonsistenzen oder Fehler erkennen. Auch wird aus einigen der Modelle durch entsprechende Werkzeuge Code generiert.

Der Ansatz der modellbasierten Entwicklung kann auch für den Test genutzt werden. Aus den Anforderungen werden nicht direkt Testfälle abgeleitet, sondern aus der Testspezifikation werden Testmodelle erstellt. Die Modelle sind dann Grundlage für die Generierung und Implementierung der Testfälle. Die Testmodelle können u.a. auf Konsistenz geprüft werden, bevor aus ihnen Testfälle generiert werden. Diese frühzeitige Prüfung hilft, Fehler zu erkennen, bevor weitere »fehlerbehaftete« Schritte durchgeführt werden.

Ein sehr großer Vorteil ist durch die automatische Generierung der Testfälle aus den Modellen gegeben: Änderungen an den Modellen führen direkt zu Änderungen der entsprechenden Testfälle.

… explorativem Test?

Exploratives Testen ist ein erfahrungsbasierter Testansatz, bei dem, wie der Name bereits andeutet, die zu testende Software zu erforschen bzw. zu erkunden ist. Es werden keine systematischen Ansätze zur Erstellung der Testfälle herangezogen. Die Kernidee des explorativen Testens ist, aus der Durchführung von Testfällen zu lernen und somit neue Ideen für weitere Testfälle zu erlangen.

Das Erkunden des Testobjekts erfolgt anhand einer sogenannten Test-Charta, die ein bestimmtes Ziel vorgibt. Zur Erkundung wird dann ein begrenzter Zeitrahmen zur Verfügung gestellt und die Ergebnisse der Testläufe werden dokumentiert.

Das Vorgehen kommt ohne große Vorbereitung aus, ist besonders bei fehlender oder veralteter Dokumentation des Testobjekts einsetzbar und

bringt Fehler zutage, die von den systematischen Testansätzen nicht immer aufgedeckt werden. Dies gilt allerdings auch umgekehrt. Daher ist exploratives Testen als Ergänzung zu den oben beschriebenen Verfahren sehr gut geeignet.

… testgetriebener Entwicklung?

Testgetriebene Entwicklung (Test-Driven Development) bezeichnet ein Vorgehen, bei dem zuerst die Testfälle auf Basis der Spezifikation entwickelt werden und erst danach der damit zu testende Programmcode geschrieben wird.

Das Vorgehen legt die Schnittstelle des zu entwickelnden Programmteils durch die Testfälle fest und ist daher eine sehr gute Designmethode, aber kein Testverfahren. Wenn bei der testgetriebenen Entwicklung die Testfälle unter Verwendung von Testverfahren erstellt werden, dann führt dies meist zu einer erheblichen Qualitätssteigerung der entwickelten Software.

… automatisierten Unit Tests?

Jede Entwicklerin prüft das von ihr entwickelte Programmstück durch automatisierte Testfälle, bevor sie es eincheckt oder freigibt. In aller Regel wird dazu ein auf die verwendete Programmiersprache zugeschnittenes Testframework genutzt. JUnit für Java ist wohl das bekannteste und auch meistgenutzte Framework. Definition, Ausführung und Auswertung der Testfälle bzw. Testsuiten werden durch das Framework unterstützt und erheblich erleichtert. Eine automatische Ausführung mit Auswertung der Testfälle und eine einfache Wiederholbarkeit sind gegeben.

Ein entsprechendes Framework sollte für alle Entwickler inzwischen zum Standardwerkzeug bei der Durchführung der Entwicklertests gehören.

… Automatisierung und Werkzeuge?

Softwareentwicklung und Testen »funktionieren« nur mit dem Einsatz von Werkzeugen. Generell muss das Testen mit Werkzeugen unterstützt werden, allerdings werden die erreichbaren Ziele durch die Automatisierung und den Einsatz von Werkzeugen oft überschätzt. Dorothy Graham hat es einmal sehr treffend formuliert: »Automating chaos just gives faster chaos.« Vor dem Einsatz der Werkzeuge muss der Testprozess eine ausreichende Reife aufweisen. Aber Testen ohne Werkzeuge ist unbefrie-

digend, jedenfalls lassen sich nicht mehrere Hundert Testfälle effizient »zu Fuß« ausführen.

Die Anzahl an Werkzeugen, die den Testprozess und das Testen unterstützen, ist sehr groß. Die Qualität und Benutzbarkeit der Werkzeuge haben in den letzten Jahren erheblich zugenommen. Neben den kommerziell verfügbaren Werkzeugen gibt es eine Reihe von Open-Source-Tools, die für den Entwicklertest, den System- und Abnahmetest oder für die Fehlerverwaltung genutzt werden können, um nur einige wenige beispielhaft zu nennen.

… künstlicher Intelligenz?

»Wie sind Systeme zu testen, die künstliche Intelligenz verwenden?« und »Wie kann künstliche Intelligenz zur Erstellung von Testfällen genutzt werden?« sind zwei Fragestellungen, die in Forschung und Praxis aktuell diskutiert werden und zu denen erste Antworten vorliegen. So gibt es seit Ende 2021 einen ISTQB®-Lehrplan (Certified Tester AI Testing) zu diesem Themenbereich.

… dem Berufsbild Testen

»Certified Tester« – das Aus- und Weiterbildungsschema des ISTQB® (International Software Testing Qualifications Board) – ist sicherlich vielen Lesern bereits bekannt. 2022 gibt es weltweit über eine 3/4 Million Zertifikate, davon in Deutschland über 100.000.

Das Aus- und Weiterbildungsportfolio umfasst inzwischen eine ganze Reihe von Grundlagen und Vertiefungen, was sowohl Testvorgehen, Testverfahren als auch Branchenspezifika betrifft.

Es gibt auch nicht mehr »das Testen«! Auch hier hat sich eine Differenzierung etabliert. Die folgende Aufzählung soll einen kleinen Eindruck vermitteln: Test-Analysis, Test-Engineering und Test-Management sowie die Spezialisierungen Testdatenmanagement, Testarchitektur, Testumgebungskoordination, Testautomatisierung, Spezialtests (Usability etc.) und Teststrategie (aus der GTB-Broschüre »Berufsbild Testen«, in ihr finden sich auch genauere Beschreibungen der einzelnen Berufsbilder).

Schlussbemerkung

Diese Broschüre kann nur Anregungen und einen ersten Einstieg geben, die hoffentlich zum weiteren Nachdenken und auch Ändern und Verbessern des gegenwärtigen Testens beitragen werden.

Dabei ist es sehr wichtig, dass Entwicklung und Test nicht als Gegenpole gesehen werden; vielmehr führt ein Miteinander von Entwicklerinnen und Testern im Team zum Erfolg. Tester müssen über Kenntnisse in der Entwicklung von Software verfügen, und Entwicklerinnen sollen auch Grundfertigkeiten im Testen besitzen. M. A. Hennell schreibt bereits 1991 hierzu[1]:

»Programmers should be educated ... in the use of testing tools and techniques. They need to see how the quality of their work can be improved without the loss of creativity and personal skill.«

In cross-funktionalen Teams der agilen Entwicklung wird versucht, diesem Anspruch ein Stück weit gerecht zu werden. In diesem Sinne: Viel Erfolg und auch Spaß beim Testen – denn Testen ist kein stupides stundenlanges Eintippen von Testdaten und Nachsehen, was »rausgekommen« ist. Es ist vielmehr eine sehr kreative und herausfordernde Tätigkeit, die direkt zur Qualitätsverbesserung beiträgt.

Wir können viel erreichen, lasst uns anfangen bzw. weitermachen und besser werden!

1. M. A. Hennell: How to Avoid Systematic Software Testing. Journal of Software Testing, Verification and Reliability, Vol. 1, No. 1, S. 23-30 (Zitat Seite 29).

Hinweise

Literatur

Manfred Baumgartner, Stefan Gwihs, Richard Seidl, Thomas Steirer, Marc-Florian Wendland: Basiswissen Testautomatisierung – Konzepte, Methoden und Techniken. 3. Auflage. dpunkt.verlag, Januar 2021.

Matthias Daigl, Rolf Glunz: ISO 29119 – Die Softwaretest-Normen verstehen und anwenden. dpunkt.verlag, Januar 2016.

Klaus Franz, Tanja Tremmel, Eckehard Kruse: Basiswissen Testdatenmanagement – Aus- und Weiterbildung zum Test Data Specialist – Certified Tester Foundation Level nach GTB. dpunkt.verlag, März 2018.

Elisabeth Hendrickson: Explore It! Wie Softwareentwickler und Tester mit explorativem Testen Risiken reduzieren und Fehler aufdecken. dpunkt.verlag, Mai 2014.

Tilo Linz: Testen in Scrum-Projekten – Leitfaden für Softwarequalität in der agilen Welt – Aus- und Weiterbildung zum ISTQB® Certified Agile Tester – Foundation Extension. 2. Auflage. dpunkt.verlag, Oktober 2016.

Frank Simon, Jürgen Grossmann, Christian Alexander Graf, Jürgen Mottok, Martin A. Schneider: Basiswissen Sicherheitstests – Aus- und Weiterbildung zum ISTQB® Advanced Level Specialist – Certified Security Tester. dpunkt.verlag, Juni 2019.

Andreas Spillner, Ulrich Breymann: Lean Testing für C++-Programmierer – Angemessen statt aufwendig testen. dpunkt.verlag, Mai 2016.

Andreas Spillner, Tilo Linz: Basiswissen Softwaretest – Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB®-Standard. 6. Auflage. dpunkt.verlag, Juni 2019.

Andreas Spillner, Mario Winter, Andrej Pietschker (Hrsg.): Test, Analyse und Verifikation von Software – gestern, heute, morgen. dpunkt.verlag, November 2017.

Michael Tamm: JUnit-Profiwissen – Effizientes Arbeiten mit der Standardbibliothek für automatisierte Tests in Java. dpunkt.verlag, Oktober 2013.

Mario Winter, Thomas Roßner, Christian Brandes, Helmut Götz: Basiswissen modellbasierter Test – Aus- und Weiterbildung zum ISTQB® Foundation Level – Certified Model-Based Tester. 2. Auflage. dpunkt.verlag, September 2016.

Erwähnte Werkzeuge

https://products.expleogroup.com/de/testona/
Testona, Werkzeug für die Klassifikationsbaum-Methode

https://csrc.nist.gov/Projects/Automated-Combinatorial-Testing-for-Software/Downloadable-Tools
Advanced Combinatorial Testing System (ACTS), Werkzeug für das kombinatorische Testen

http://junit.org/
Frameworks für Unit Tests für die Programmiersprache Java (ähnliche Frameworks gibt es inzwischen für nahezu alle Programmiersprachen)

Internetseiten

http://www.softwaretest-knowledge.de/
Seite zum Buch »Basiswissen Softwaretest«

http://softwaretest-umfrage.de/
Ergebnisse der Umfrage »Softwaretest in Praxis und Forschung« 2020

http://www.istqb.org/
International Software Testing Qualifications Board

http://german-testing-board.info/
German Testing Board e.V.

http://softwaretestingstandard.org/
ISO/IEC/IEEE 29119 Software Testing

http://leantesting.de/
Seite zum Buch »Lean Testing für C++-Programmierer«, Links zu Testwerkzeugen (für C++ u.a.)
http://leantesting.de/Openbook_Testen.pdf
Openbook – Kombinationen mit n-weisem Testen und Entscheidungstabellen

https://www.asqf.de/
Arbeitskreis Software-Qualität und -Fortbildung e.V.

http://fg-tav.gi.de/
Fachgruppe »Test, Analyse und Verifikation von Software« der Gesellschaft für Informatik e.V.

GTB

German Testing Board

2002 – 2022

20 Jahre Software. Testing. Excellence.

Wir sind seit 20 Jahren Expert:innen für die praxisrelevante Qualifizierung von Software-Tester:innen und schaffen einen Industriestandard für Software-Testing Fähigkeiten.

GTB Premium Partner

CGI · imbus · knowledge department · loyal team · Lysant

methodpark · oose. · QUALITYDOJO · sepp.med

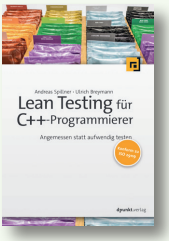
software quality lab · sogeti · T-Systems · T-Systems

www.gtb.de

Andreas Spillner · Tilo Linz

Basiswissen Softwaretest

6., überarb. und akt. Auflage
378 Seiten, Festeinband
€ 39,90 (D)
ISBN 978-3-86490-583-4



Andreas Spillner
Ulrich Breymann

Lean Testing für C++-Programmierer

246 Seiten, Broschur
€ 29,90 (D)
ISBN 978-3-86490-308-3

Florian Fieber · Marc-Florian Wendland

Basiswissen Abnahmetest

242 Seiten, Festeinband
€ 32,90 (D)
ISBN 978-3-86490-829-3

Björn Lemke · Nils Röttger

Basiswissen Mobile App Testing

200 Seiten, Festeinband
€ 22,90 (D)
ISBN 978-3-86490-748-7

Ralf Bongard · Klaudia Dussa-Zieger · Ralf Reißing · Alexander Schulz

Basiswissen Automotive Softwaretest

252 Seiten, Festeinband
€ 34,90 (D)
ISBN 978-3-86490-580-3

Manfred Baumgartner · Stefan Gwihs · Richard Seidl · Thomas Steirer · Marc-Florian Wendland

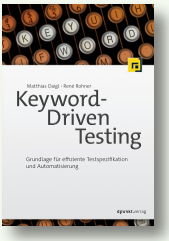
Basiswissen Testautomatisierung

3., akt. und überarb. Auflage
398 Seiten, Festeinband
€ 39,90 (D)
ISBN 978-3-86490-675-6

Frank Simon · Jürgen Grossmann · Christian Alexander Graf · Jürgen Mottok · Martin A. Schneider

Basiswissen Sicherheitstests

414 Seiten, Festeinband
€ 39,90 (D)
ISBN 978-3-86490-618-3



Matthias Daigl · René Rohner

Keyword-Driven Testing

260 Seiten, Festeinband
€ 34,90 (D)
ISBN 978-3-86490-570-4

dpunkt.verlag GmbH · Wieblinger Weg 17 · 69123 Heidelberg · www.dpunkt.de · bestellung@dpunkt.de

Mission Softwarequalität

Beratung	Bessere Software in kürzerer Zeit
Softwaretest	Fehler schnell und sicher finden
Akademie	Lernen wann und wo ich will
TestBench	Die smarte Test Management Lösung

imbus sichert die Qualität und die Funktionalität von softwarebasierten Produkten für eine digitalisierte Zukunft!

www.imbus.de

Andreas Spillner

Systematisches Testen von Software

Das Testen von Software erfordert in der Praxis einen erheblichen Aufwand. Dieser ist erst dann gerechtfertigt, wenn das Testen auch die gewünschten Nachweise liefert: Die Software hat die geforderte Qualität und enthält keine gravierenden Fehler.

Wird unstrukturiert und »ad hoc« getestet (oder präziser formuliert: ausprobiert), lässt sich dieser Nachweis kaum erbringen oder gar fundiert belegen. Um eine systematische Prüfung der Software vorzunehmen, wird die Anwendung von Testverfahren zur Erstellung der Testfälle empfohlen. Durch die Anwendung der Verfahren ist sichergestellt, dass keine wichtigen Testfälle übersehen werden. Darüber hinaus geben die Testverfahren Kriterien an die Hand, wann das Testen als ausreichend angesehen und beendet werden kann.

Beispielhaft werden spezifikations- und strukturbasierte Testverfahren vorgestellt. Auf den aktuellen Standard »ISO/IEC/IEEE 29119 Software Testing« wird ebenso eingegangen wie auf das Aus- und Weiterbildungsschema des ISTQB® (International Software Testing Qualifications Board) zum »Certified Tester«. Da zum Testen neben der Ausführung von Testfällen noch eine ganze Reihe von Tätigkeiten gehören, sind Testprozesse definiert worden. Zwei davon werden kurz beschrieben.

4., erweiterte und aktualisierte Auflage
Art.-Nr.: 077.95748
Schutzgebühr: 3,00 €

www.dpunkt.de