

2 | Funktionsorientierter Test

Im Folgenden werden dynamische Testtechniken dargestellt, welche die Vollständigkeit des Tests anhand der Abdeckung einer Spezifikation mit Testfällen beurteilen. Da Spezifikationen die Soll-Funktionalität einer Software festlegen, werden die Testtechniken, die gegen Spezifikationen testen, als funktionsorientierte Testtechniken bezeichnet. Funktionsorientierte Testtechniken sind unverzichtbar. Standards verlangen stets eine sorgfältige funktionsorientierte Testplanung. Dies ist eine direkte Forderung zur Nutzung systematischer funktionsorientierter Testtechniken. Funktionsorientiertes Testen findet in allen Testphasen statt. Im Modultest werden Modulspezifikationen als Basis für den Test herangezogen. Im Integrationstest bilden Schnittstellenspezifikationen die Testreferenz. Im Systemtest wird gegen Anforderungsdefinitionen getestet. Ein Test ohne systematische funktionsorientierte Testplanung ist als ungenügend zu betrachten.

Übersicht

2.1	Eigenschaften und Ziele des funktionsorientierten Tests	50
2.2	Funktionale Äquivalenzklassenbildung	51
2.3	Zustandsbasierter Test	58
2.4	Ursache-Wirkungs-Analyse	66
2.5	Weitere funktionsorientierte Testtechniken	73
2.6	Bewertung des funktionsorientierten Tests	81
	Checkliste	81

2.1 Eigenschaften und Ziele des funktionsorientierten Tests

Testgrundlage: Spezifikation

Funktionsorientierte Testtechniken definieren Regeln für die Testdatenerzeugung

PROBLEM: Un- getesteter Code

Die funktionsorientierten Testtechniken stellen die Software-Spezifikation in den Mittelpunkt des Testgeschehens. Es ist daher insbesondere darauf zu achten, dass eine zum jeweiligen Test passende Spezifikation zur Verfügung steht und auch verwendet wird. Abb. 2.1 zeigt das Prinzip des funktionsorientierten Tests. Der Tester liest die Spezifikation und gewinnt so ein Bild ihrer Inhalte. Dieses wird nach festen Regeln in Testfälle übersetzt. Bei der Durchführung der Testfälle entstehen Reaktionen der Software. Die Korrektheit dieser Reaktionen ist anhand des Bildes der Spezifikation zu beurteilen. Das Testende ist erreicht, wenn das Bild der Spezifikation mit Testfällen vollständig abgedeckt ist. Der Spezifikation kommt beim funktionsorientierten Testen eine zentrale Bedeutung zu. Sie dient zur Beurteilung der Vollständigkeit des Tests sowie zur Herleitung der Testfälle und der Beurteilung der Reaktionen der Software. Aufgrund ihrer Herleitung aus der Spezifikation sind die Testfälle eines funktionsorientierten Tests systematisch an der Überprüfung der Soll-Funktionalität ausgerichtet. Daher kann insbesondere geprüft werden, ob die Spezifikation vollständig in Software umgesetzt wurde. Ein erfolgreich durchgeführter funktionsorientierter Test vermittelt daher ein gesteigertes Vertrauen in die Software-Qualität. Neben den genannten Vorteilen existieren einige Nachteile. So kann ein funktionsorientierter Test z. B. nicht gewährleisten, dass der Programmcode vollständig getestet wird. Bei einem reinen funktionsorientierten Test ist daher davon auszugehen, dass in der Software noch ungetesteter Code existiert.

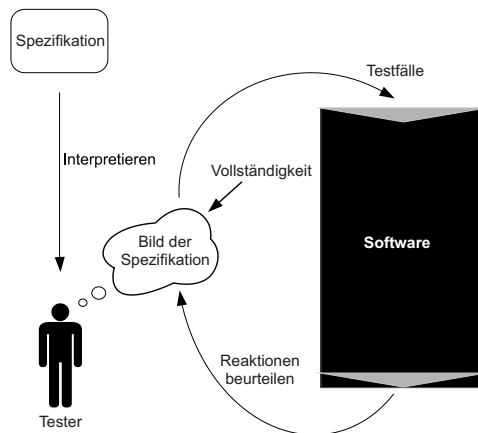


Abbildung 2.1 Prinzip des funktionsorientierten Tests

Da die Abläufe innerhalb der getesteten Software beim funktionsorientierten Testen irrelevant sind, gehören die funktionsorientierten Test-

techniken zu den so genannten *Black Box*-Tests. *Black Box*-Testen und funktionsorientiertes Testen sind jedoch keine Synonyme. Alle funktionsorientierten Testtechniken sind *Black Box*-Techniken, aber nicht alle *Black Box*-Techniken sind funktionsorientiert. Funktionsorientiertes Testen erzwingt die Erzeugung von Testfällen aus Spezifikationen. *Black Box*-Testen bedeutet, dass auf eine Betrachtung der Programmstruktur verzichtet wird. Weil funktionsorientierte Testtechniken die Testfälle anhand der Spezifikation erzeugen und die Spezifikation verwenden, um Vollständigkeit zu bewerten, können sie auf eine Betrachtung der Programmstruktur verzichten. Sie sind daher *Black Box*-Techniken. Andere *Black Box*-Techniken verzichten aus anderen Gründen auf die Betrachtung der Programmstruktur. Es handelt sich dann um *Black Box*-Techniken, die nicht funktionsorientiert testen.

Funktionsorientierter Test \neq *Black Box*-Test

Der Zufallstest (Abschnitt 5.3) ist ein Beispiel für eine *Black Box*-Technik, die nicht funktionsorientiert testet.

2.2 Funktionale Äquivalenzklassenbildung

2.2.1 Eigenschaften und Ziele der funktionalen Äquivalenzklassenbildung

Ein Hauptproblem beim dynamischen Software-Test ist die Auswahl einiger weniger Testfälle aus einer oft umfangreichen Menge von möglichen Betriebssituationen. Es ist schwierig, aus einer großen Menge potentieller Testfälle einige wenige auszuwählen, die durchgeführt werden sollen. Die Auswahl einiger Testfälle bedeutet, dass sich der Tester implizit entscheidet, eine große Menge von Betriebssituationen nicht zu testen. Daher muss die Auswahl der Testfälle sehr sorgfältig geschehen. Es ist darauf zu achten, dass die gewählten Testfälle gute Stellvertreter der Grundgesamtheit darstellen. Darüber hinaus sollen sie so beschaffen sein, dass potentiell enthaltene Fehler möglichst zuverlässig erkannt werden. Des Weiteren sollen die Testfälle frei von Redundanzen sein. Die Funktionale Äquivalenzklassenbildung nutzt ein Prinzip, das sich als Verfahren zur Beherrschung von Komplexität an vielen Stellen als nützlich erwiesen hat: das Prinzip „Teile und Herrsche“. Die funktionale Äquivalenzklassenbildung versucht die Komplexität eines Testproblems durch fortgesetztes Zerlegen so weit zu reduzieren, dass schließlich eine sehr einfache Wahl von Testfällen möglich wird. Am Ende der Zerlegung steht eine Menge von so genannten Äquivalenzklassen. Alle Werte einer Äquivalenzklasse sollen durch die zu testende Software gleichartig bearbeitet werden. Daher ist davon auszugehen, dass jeder Wert einer Äquivalenzklasse ein geeigneter repräsentativer Stellvertreter für alle Werte der Äquivalenzklasse ist. Da die Äquivalenzklassen anhand der Spezifikation identifiziert werden, heißt das Verfahren funktionale Äquivalenzklassenbildung.

**Englisch:
*Divide and conquer***

2.2.2 Beschreibung der funktionalen Äquivalenzklassenbildung

Ansatz: Fallunterscheidung

Der grundsätzliche Ansatz bei der funktionalen Äquivalenzklassenbildung ist die Durchführung einer fortgesetzten Fallunterscheidung. Dies geschieht sowohl für Eingabe- als auch für Ausgabebedingungen der zu testenden Software. Werte aus einer Äquivalenzklasse sollen ein identisches funktionales Verhalten verursachen. Sie testen insbesondere dieselbe spezifizierte Programmfunktion. Einerseits stellt die Bildung von Äquivalenzklassen anhand der Spezifikation sicher, dass alle spezifizierten Programmfunktionen mit Werten ihrer Äquivalenzklasse getestet werden. Andererseits wird eine Beschränkung der Testfallanzahl erreicht. Eingabeäquivalenzklassen werden direkt für Eingabebedingungen gebildet. Falls die Spezifikation z. B. ausschließlich positive Werte für eine bestimmte Eingabe gestattet, so können anhand dieser Forderung zwei Fälle unterschieden werden: positive Eingabewerte und nicht-positive Eingabewerte.

Gültige vs. ungültige Äquivalenzklassen

Die positiven Eingabewerte stellen den Normalfall dar. Man spricht von so genannten gültigen Äquivalenzklassen. Nicht-positive Eingaben dürfen nicht auftreten. Treten sie dennoch auf, so liegt ein Fehlerfall vor, der einer Fehlerbehandlung bedarf. Man spricht von so genannten ungültigen Äquivalenzklassen. Die Äquivalenzklassenbildung für Ausgabebedingungen erfolgt ebenfalls anhand einer Fallunterscheidung auf Basis der Spezifikation. Nachdem die Ausgabeäquivalenzklassen vorliegen müssen zusätzlich Eingabewerte ermittelt werden, die Ausgaben in den jeweils betrachteten Ausgabeäquivalenzklassen verursachen. Die Auswahl der konkreten Testdaten aus einer Äquivalenzklasse kann nach unterschiedlichen Kriterien erfolgen. Eine oft verwendete Vorgehensweise ist der Test der Äquivalenzklassengrenzen. Dieses heuristische Verfahren wird als Grenzwertanalyse bezeichnet. Es basiert auf der Erfahrung, dass Fehler besonders häufig an den Grenzen von Äquivalenzklassen auftreten. Andere mögliche Ansätze sind der Test besonderer Werte, beispielsweise der Wert 0, oder auch eine zufallsorientierte stochastische Vorgehensweise. Bereits in /Myers 79/ ist die Bildung von ungültigen und gültigen Äquivalenzklassen vorgesehen. Ist für eine Eingabe z. B. ein Wertebereich vorgesehen, so stellt dieser Bereich eine gültige Äquivalenzklasse dar, die gegebenenfalls weiter aufzutrennen ist, und die an ihrer unteren und oberen Grenze durch jeweils eine eigene ungültige Äquivalenzklasse begrenzt wird.

Grenzwertanalyse

BEISPIEL

Eingabebereich: $1 \leq \text{Wert} \leq 99$

Eine gültige Äquivalenzklasse: $1 \leq \text{Wert} \leq 99$

Zwei ungültige Äquivalenzklassen: $\text{Wert} < 1$, $\text{Wert} > 99$

Im Einzelnen werden folgende Regeln zur Äquivalenzklassenbildung benutzt /Myers 79/:

Falls eine Eingabebedingung einen Wertebereich spezifiziert, so sind eine gültige Äquivalenzklasse und zwei ungültige Äquivalenzklassen zu bilden (siehe oben angegebenes Beispiel).

Spezifiziert eine Eingabebedingung eine Anzahl von Werten, so sind eine gültige Äquivalenzklasse und zwei ungültige Äquivalenzklassen zu bilden.

Für ein Auto können zwischen einem und sechs Besitzer eingetragen sein.

Eine gültige Äquivalenzklasse:

- > Ein Besitzer bis sechs Besitzer

Zwei ungültige Äquivalenzklassen:

- > Kein Besitzer
- > Mehr als sechs Besitzer

BEISPIEL

Falls eine Eingabebedingung eine Menge von Werten spezifiziert, die unterschiedlich behandelt werden, so ist für jeden Wert eine eigene gültige Äquivalenzklasse zu bilden. Für alle Werte mit Ausnahme der gültigen Werte ist eine ungültige Äquivalenzklasse zu bilden.

Tasteninstrumente: Klavier, Cembalo, Spinett, Orgel.

Vier gültige Äquivalenzklassen:

- > Klavier
- > Cembalo
- > Spinett
- > Orgel

Eine ungültige Äquivalenzklasse:

- > Alles andere, z. B. Violine

BEISPIEL

Falls eine Eingabebedingung eine Situation festlegt, die zwingend erfüllt sein muss, so sind eine gültige Äquivalenzklasse und eine ungültige Äquivalenzklasse zu bilden.

BEISPIEL

Das erste Zeichen muss ein Buchstabe sein.

Eine gültige Äquivalenzklasse:

- Das erste Zeichen ist ein Buchstabe.

Eine ungültige Äquivalenzklasse:

- Das erste Zeichen ist kein Buchstabe (z. B. Ziffer oder Sonderzeichen).

Falls Grund zu der Annahme besteht, dass Elemente einer Äquivalenzklasse unterschiedlich behandelt werden, so ist diese Äquivalenzklasse entsprechend aufzutrennen.

Bildung von Ausgabe- äquivalenzklassen

Diese Regeln für die Bildung von Eingabeäquivalenzklassen können auf die Bildung von Ausgabeäquivalenzklassen übertragen werden. Als Beispiel ist hier die Regel für die Situation, dass ein Ausgabewertebereich spezifiziert ist, angegeben. Spezifiziert eine Ausgabebedingung einen Wertebereich, so sind alle Eingabewerte, die Ausgaben innerhalb des Wertebereichs erzeugen, einer gültigen Äquivalenzklasse zuzuordnen. Alle Eingaben, die Ausgaben unterhalb des spezifizierten Wertebereichs verursachen, werden einer ungültigen Äquivalenzklasse zugeordnet. Alle Eingaben, die Ausgaben oberhalb des spezifizierten Wertebereichs verursachen, werden einer anderen ungültigen Äquivalenzklasse zugeordnet.

BEISPIEL

Ausgabebereich: $1 \leq \text{Wert} \leq 99$

Eine gültige Äquivalenzklasse:

- Alle Eingaben, die Ausgaben zwischen 1 und 99 erzeugen.

Zwei ungültige Äquivalenzklassen:

- Alle Eingaben, die Ausgaben kleiner als 1 erzeugen
- Alle Eingaben, die Ausgaben größer als 99 erzeugen

Die Bildung gültiger und ungültiger Äquivalenzklassen ist sinnvoll für Programme, die ihre Eingaben über Ein-/Ausgabekanäle erhalten, da für ungültige Äquivalenzklassen Fehlerbehandlungen existieren müssen. Für unterlagerte Operationen, die ihre Eingaben über Parameterschnittstellen erhalten, sind die Beschränkungen bezüglich der Eingaben zu beachten. Das aufrufende Programm führt möglicherweise gewisse Fehlerbehandlungen bereits durch. Eine erneute Fehlerabfrage in der Operation

führt in diesem Fall zu dynamisch nicht erreichbar Code. Oft existieren auch bestimmte Abhängigkeiten zwischen Eingaben, die eine Herstellung mancher Eingabekonstellationen ausschließen. Durch Nichtbeachtung dieser Aspekte werden Testfälle erzeugt, die für den Test ungeeignet sind.

Für die Operation *ZaehleZchn* können die folgenden Äquivalenzklassen gebildet werden (siehe Beispielspezifikation in Abschnitt 2.4):

1. $Zchn < 'A'$
2. $Zchn > 'Z'$
3. $Zchn \geq 'A'$ und $Zchn \leq 'Z'$
4. Zchn ist ein großer Konsonant.
5. Zchn ist ein großer Vokal.
6. Zchn ist weder ein großer Konsonant noch ein großer Vokal.
7. $Gesamtzahl < INT_MAX$
8. $Gesamtzahl = INT_MAX$
9. $Gesamtzahl > INT_MAX$

Für *VokalAnzahl* müssen keine Äquivalenzklassen gebildet werden, da aufgrund der Abhängigkeit zwischen *VokalAnzahl* und *Gesamtzahl* die Verarbeitung vom Wert der Variablen *VokalAnzahl* unabhängig ist.

Die Äquivalenzklassen sind zum Teil nicht disjunkt. So ergeben die Äquivalenzklassen 4 und 5 zusammen die Äquivalenzklasse 3. Außerdem sind nicht zu allen Äquivalenzklassen Testfälle erzeugbar. Aufgrund des Datentyps von *Gesamtzahl* können Testfälle zur Äquivalenzklasse 9 nicht erzeugt werden.

BEISPIEL

Die Äquivalenzklassen sind eindeutig zu nummerieren. Für die Erzeugung von Testfällen aus den Äquivalenzklassen sind zwei Regeln zu beachten:

- Die Testfälle für gültige Äquivalenzklassen werden durch Auswahl von Testdaten aus möglichst vielen gültigen Äquivalenzklassen gebildet. Dies reduziert die Testfälle für gültige Äquivalenzklassen auf ein Minimum.
- Die Testfälle für ungültige Äquivalenzklassen werden durch Auswahl eines Testdatums aus einer ungültigen Äquivalenzklasse gebildet. Es wird mit Werten kombiniert, die ausschließlich aus gültigen Äquivalenzklassen entnommen sind. Da für alle ungültigen Eingabewerte eine Fehlerbehandlung existieren muss, kann bei Eingabe eines fehlerhaften Wertes pro Testfall die Fehlerbehandlung nur durch

Bildung von Testfällen aus Äquivalenzklassen

dieses fehlerhafte Testdatum verursacht worden sein. Würden mehrere fehlerhafte Eingaben pro Testfall verwendet, so ist nicht erkennbar, welches fehlerhafte Testdatum die Fehlerbehandlung ausgelöst hat.

BEISPIEL

Ein Programm zur Lagerverwaltung einer Baustoffhandlung besitzt eine Eingabemöglichkeit für die Registrierung von Anlieferungen. Eine Holzbretter angeliefert, so wird die Holzart eingegeben. Das Programm kennt die Holzarten Eiche, Buche und Kiefer. Ferner wird die Länge in Zentimetern angegeben, die stets zwischen 100 und 500 liegt. Als gelieferte Anzahl kann ein Wert zwischen 1 und 9999 angegeben werden. Außerdem erhält die Lieferung eine Auftragsnummer. Jede Auftragsnummer für Holzlieferungen beginnt mit dem Buchstaben H.

Anhand dieser Beschreibung können, unter Verwendung der angegebenen Regeln, die Äquivalenzklassen nach Tab. 2.1 gebildet werden. Tab. 2.2 stellt einen vollständigen Satz von Testfällen dar. Die Testdaten sind ohne Anwendung einer konkreten Strategie aus den Äquivalenzklassen gewählt.

Die in Tab. 2.3 dargestellten Testfälle sind durch gemeinsame Anwendung von Äquivalenzklassenanalyse und Grenzwertanalyse entstanden. Als Testdaten sind systematisch die Grenzen der Äquivalenzklassen gewählt worden. Das Kürzel U oder O hinter der Angabe der getesteten Äquivalenzklasse kennzeichnet einen Test der unteren bzw. oberen Grenze der angegebenen Äquivalenzklasse.

Eingabe	gültige Äquivalenzklassen	ungültige Äquivalenzklassen
Holzart	1) Eiche 2) Buche 3) Kiefer	4) Alles andere, z. B. Stahl
Länge	5) $100 \leq \text{Länge} \leq 500$	6) Länge < 100 7) Länge > 500
Anzahl	8) $1 \leq \text{Anzahl} \leq 9999$	9) Anzahl < 1 10) Anzahl > 9999
Auftragsnummer	11) Erstes Zeichen ist H	12) Erstes Zeichen ist nicht H

Tabelle 2.1 Äquivalenzklassen-Aufstellung

Testfall	1	2	3	4	5	6	7	8	9
(zusätzlich) getestete Äquivalenz- klassen	1) 5) 8) 11)	2)	3)	4)	6)	7)	9)	10)	12)
Holzart	Eiche	Buche	Kiefer	Stahl	Buche	Buche	Buche	Buche	Buche
Länge	200	300	300	300	50	1000	200	200	200
Anzahl	100	200	100	100	100	100	0	50000	100
Auftrags- nummer	H1	H1	H1	H1	H1	H1	H1	H1	J2

Tabelle 2.2 Testfälle nach einer Äquivalenzklassenanalyse

Testfall	1	2	3	4	5	6	7	8	9
(zusätzlich getestete Äquivalenz- klassen	1) U 5) U 8) U 11)	2) O 5) O 8) O	3)	4)	6) O	7) U	9) O	10) U	12)
Holzart	Eiche	Buche	Kiefer	Stahl	Buche	Buche	Buche	Buche	Buche
Länge	100	500	300	300	99	501	200	200	200
Anzahl	1	9999	100	100	100	100	0	10000	100
Auftrags- nummer	H1	H1	H1	H1	H1	H1	H1	H1	J2

Tabelle 2.3 Testfälle nach einer Äquivalenzklassenanalyse und Grenzwert-analyse

2.2.3 Bewertung der funktionalen Äquivalenzklassenbildung

Die funktionale Äquivalenzklassenbildung ist eine ziemlich universell einsetzbare Testtechnik. Im Modultest wird eine Fallunterscheidung bezüglich konkreter Ein- und Ausgabewerte durchgeführt. Im Integrations-test werden die über Schnittstellen möglichen Interaktionen unterschieden. Im Systemtest werden unterschiedliche Anwendungsfälle unterschieden. Das Verfahren ist einfach anwendbar. Darüber hinaus ist zu erwarten, dass zahlreiche Softwaretester, die bislang ohne konkrete Methodik gearbeitet haben, implizit bereits ähnliche Verfahren angewendet haben. Die Durchführung einer funktionalen Äquivalenzklassenbildung in der hier beschriebenen Vorgehensweise systematisiert daher sicherlich in vielen Fällen nur eine Technik, die in einfacherer Form bereits etabliert ist. Es ist deshalb damit zu rechnen, dass die Einführung der funktionalen Äquivalenzklassenbildung als Testtechnik problemlos von-statten gehen wird.

Aufgrund der beschriebenen Vorgehensweise, die vorsieht, dass Testfälle tabellarisch geplant und abgelegt werden, ist zu erwarten, dass im Falle des Wechsels von Testzuständigkeiten die Einarbeitung erleichtert wird. Darüber hinaus ist es möglich, Fehler, die beim Testen nicht gefunden und zu einem späteren Zeitpunkt erkannt wurden, auf die Äquivalenzklassenschemata abzubilden, um die Äquivalenzklassenbildung zu optimieren. Neben den genannten Vorteilen existieren einige Nachteile. Die funktionale Äquivalenzklassenbildung gestattet es nicht, Wechselwirkungen zwischen Äquivalenzklassen zu beschreiben. Falls Software-Reaktionen an eine ganz bestimmte Verknüpfung von Äquivalenzklassen gebunden sind, so kann dies bei der funktionalen Äquivalenzklassenbildung nicht geeignet beschrieben werden. Die im Folgenden noch beschriebene Ursache-Wirkungs-Analyse bietet eine Lösungsmöglichkeit für dieses Problem. Die funktionale Äquivalenzklassenbildung bereitet weiterhin Schwierigkeiten bei der Anwendung auf zustandsbasierte Soft-

Vorteil: Einfachheit

Gute Testdokumentation erleichtert die Einarbeitung

PROBLEME: Wechselwirkungen von Äquivalenzklassen sind nicht erfassbar; zustandsbasierte Software ist nicht behandelbar.

ware. In zustandsbasierter Software wird im Allgemeinen auf Eingaben je nach Zustand der Software unterschiedlich reagiert. Diese Abhängigkeit zwischen Äquivalenzklassen und Zuständen kann ebenfalls nicht geeignet beschrieben werden.

Zusammenfassend kann formuliert werden: Die funktionale Äquivalenzklassenbildung ist gut geeignet für den Test von Software, die nicht zustandsbasiert ist und bei der die Reaktionen nicht abhängig sind von komplizierten Eingabeverknüpfungen.

2.3 Zustandsbasierter Test

2.3.1 Eigenschaften und Ziele des zustandsbasierten Tests

Die Unterscheidung zwischen zustandsbasierten und zustandsfreien Systemen ist auch in anderen Disziplinen als der Informatik verbreitet. So unterscheidet man z. B. in der Elektrotechnik Schaltnetze – zustandsfreie Schaltungen – und Schaltwerke – zustandsbehaftete Schaltungen. Man spricht insgesamt auch von gedächtnisbehafteten und gedächtnislosen Systemen. Streng genommen ist jede Software gedächtnisbehaftet, denn sie verfügt über einen Speicher. Diese Gedächtnisbehaftung steht aber nicht stets so im Vordergrund, dass sie für den Test beachtet werden muss. Falls sie ignoriert werden kann, so reicht eine funktionale Äquivalenzklassenbildung als Testtechnik aus. Falls das Gedächtnis das Verhalten der Software wesentlich bestimmt, so muss eine Testtechnik verwendet werden, die dies berücksichtigen kann. Der zustandsbasierte Test ist eine solche Technik.

Zustandsbasierte vs. zustandsfreie Systeme

Zustandsautomaten sind ein verbreitetes Beschreibungsmittel.

Zustandsautomaten sind ein verbreitetes Beschreibungsmittel in technischen Disziplinen. In der Informatik sind sie eine so genannte Basistechnik zur Beschreibung von Verhalten. Moderne Analyse- und Entwurfstechniken – z. B. UML (*Unified Modeling Language*) – bieten Zustandsautomaten als Beschreibungsmittel an. In der UML können Zustandsautomaten zur Beschreibung des Verhaltens von Klassen genutzt werden. Zustandsbasiertes Testen ist eine funktionsorientierte Testtechnik für Software, die als Zustandsautomat spezifiziert ist oder zumindest als Zustandsautomat spezifiziert werden kann.

2.3.2 Beschreibung des zustandsbasierten Tests

Zustandsbasiertes Testen zielt auf die vollständige Testabdeckung von Zustandsautomaten. Da die Zustandsautomaten eine Spezifikation sind, handelt es sich hier um einen funktionsorientierten Ansatz. Im Folgenden ist ein Ausschnitt einer Modulspezifikation in Textform angegeben. Die Modulspezifikation besitzt einige Unzulänglichkeiten, die typisch für textuelle Beschreibungen von zustandslastiger Software sind.

Software-Qualität

Testen, Analysieren und Verifizieren von Software

Liggesmeyer, P.

2009, XV, 526 S., Hardcover

ISBN: 978-3-8274-2056-5