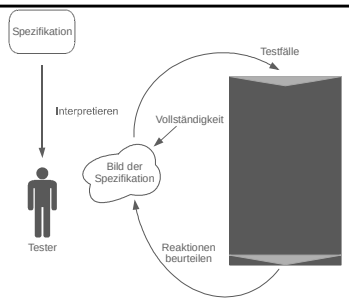
Funktionsorientierter Test

- Beurteilung der Adäquanz und der Vollständigkeit des Tests, sowie Herleitung der Testdaten und Beurteilung der Ausgaben anhand der Modulspezifikation.
  - Vorteil: Vollständigkeit der Umsetzung der Spezifikation wird geprüft
  - Nachteil: Struktur der Implementation wird nicht berücksichtigt
- Hier vorgestellte Techniken:
  - Funktionale Äquivalenzklassenbildung
  - Zustandsautomaten-orientierter Test

Funktionsorientierter Test
Arbeitsweise



Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse

- Bildung von Äquivalenzklassen der Eingabewerte auf der Basis der funktionalen Eigenschaften des Programms oder besser seiner funktionalen Spezifikation.
- Werte aus einer Äquivalenzklasse
  - verursachen ein identisches funktionales Verhalten und
  - testen eine identische spezifizierte Programmfunktion.
- Die Bildung der Äquivalenzklassen anhand der Spezifikation stellt sicher, dass alle spezifizierten Programmfunktionen mit Werten aus der ihnen zugeordneten Äquivalenzklasse getestet werden.
- Aus den Ausgabewertebereichen können ebenfalls Äquivalenz-klassen erstellt werden.

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse: Bildung von ungültigen und gültigen Äquivalenzklassen

- Beispiel:
  - Ist für eine Eingabe ein Wertebereich vorgesehen, so stellt dieser Bereich eine gültige Äquivalenzklasse dar, die an ihrer unteren und oberen Grenze durch ungültige Äquivalenzklassen eingerahmt wird.
  - Eingabebereich: $1 \leq \text{Wert} \leq 99$
  - Eine gültige Äquivalenzklasse: $1 \leq \text{Wert} \leq 99$
  - Zwei ungültige Äquivalenzklassen: $\text{Wert} < 1$, $\text{Wert} > 99$

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse:
Bildung von ungültigen und gültigen Äquivalenzklassen

- 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.
  - 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.
- Auswahl der konkreten Testdaten aus einer Äquivalenzklasse nach unterschiedlichen Kriterien.
- Oft verwendet: Test der Äquivalenzklassengrenzen **(Grenzwertanalyse)**

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse:
Bildung von ungültigen und gültigen Äquivalenzklassen

- Beispiel
  - Ein Programm zur Lagerverwaltung einer Baustoffhandlung besitzt eine Eingabemöglichkeit für die Registrierung von Anlieferungen.
  - Werden 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.

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse:
Bildung von ungültigen und gültigen Äquivalenzklassen

Äquivalenzklassenaufstellung

Eingabe	gültige Äquivalenzklassen	ungültige Äquivalenzklassen
Holzart	1) Eiche 2) Buche 3) Kiefer	4) Stahl
Länge	5) 100 ≤ Länge ≤ 500	6) Länge < 100 7) Länge > 500
Anzahl	8) 1 ≤ Anzahl ≤ 9999	9) Anzahl < 1 10) Anzahl > 9999
Auftrags-nummer	11) Erstes Zeichen ist *H*	12) Erstes Zeichen ist nicht *H*

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse:
Bildung von ungültigen und gültigen Äquivalenzklassen

Testfälle nach Äquivalenzklassenbildung mit Grenzwertanalyse

Testfall	1	2	3	4	5	6	7	8	9
getestete Äquivalenz-klassen	1) 5) U 8) U 11)	2) 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**

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse

Beispiel

Die Klasse "Dreieck" enthält als Attribute die Längen der Dreiecksseiten Seite1, Seite2 und Seite3 als ganze Zahlen. Die Operation "Art ()" ermittelt die Dreiecksart auf Basis dieser Seitenlängen. Es werden folgende Fälle unterschieden:

- Kein Dreieck: Datenfehler der Seitenlängen,
- Gleichseitig,
- Rechtwinklig,
- Gleichschenklig,
- Ungleichseitig.

Die Art Rechtwinklig wird mit Vorrang ausgegeben, d.h. falls z.B. ein ungleichseitiges Dreieck rechtwinklig ist, so wird nicht Ungleichseitig, sondern Rechtwinklig ausgegeben.

Funktionsorientierter Test
Funktionale Äquivalenzklassenanalyse

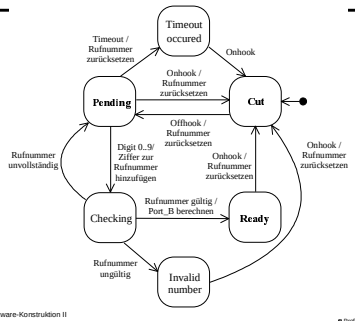
Äquivalenzklassen und Testfälle

Bedingung	gültig	ungültig	Testfall
Seite 1 (S1)	>0	≤ 0	0, 1, 1
Seite 2 (S2)	>0	≤ 0	1, 0, 1
Seite 3 (S3)	>0	≤ 0	1, 1, 0
KeinDreieck	S1 + S2 ≤ S3		1, 1, 2
	S1 + S3 ≤ S2		1, 2, 1
	S2 + S3 ≤ S1		2, 1, 1
Gleichseitig	S1 = S2 = S3		1, 1, 1
Gleichschenklig	S1 = S2 ≠ S3		1, 1, 3
	S1 = S3 ≠ S2		1, 3, 1
	S2 = S3 ≠ S1		3, 1, 1
Rechtwinklig	$S1^2 + S2^2 = S3^2$		3, 4, 5
	$S1^2 + S3^2 = S2^2$		3, 5, 4
	$S2^2 + S3^2 = S1^2$		5, 3, 4
Ungleichseitig	S1 ≠ S2 ≠ S3 und nicht rechtwinklig		2, 3, 4

Funktionsorientierter Test
Zustandsbasierter Test

- Anwendungsgebiet: Software, die einen Zustandsautomaten realisiert
- Beispiel: Ausschnitt einer Modul-Spezifikation:
  - Parameter:
    - PORT_A: Wählender Teilnehmeranschluß
    - PORT_B: Teilnehmeranschluß des angewählten Gesprächspartners
  - PORT_A identifiziert den Teilnehmeranschluß, von dem aus ein Gespräch aufgebaut werden soll. Der aktuelle Zustand des Gesprächsaufbaus steht global zur Verfügung. Abhängig davon ergibt sich nach Auswertung der übergebenen Aktion ein neuer Zustand. Der zurückgelieferte Zustand ist CUT, falls der Gesprächsaufbau abgebrochen wurde, er ist PENDING falls der Gesprächsaufbau im Gange aber noch nicht beendet ist. Er ist READY, falls der Gesprächsaufbau erfolgreich abgeschlossen wurde. In diesem Fall liefert PORT_B den Teilnehmeranschluß des angewählten Teilnehmers, sonst ist der Inhalt von PORT_B undefiniert. Ein Gesprächsaufbau erfordert die Sequenz OFFHOOK (DIGIT_N)*, wobei die eingegebenen Ziffernfolge eine gültige Rufnummer darstellt. ONHOOK führt immer zum vollständigen Abbruch des Gesprächs.Falls TIMEOUT auftritt, so kann nur durch Auflegen (ONHOOK) in den Grundzustand (CUT) zurückgekehrt werden.

Funktionsorientierter Test
Zustandsbasierter Test



Funktionsorientierter Test
Zustandsbasierter Test

- Eine minimale Teststrategie ist die mindestens einmalige Abdeckung aller Zustände durch Testfälle.
- Besser ist das mindestens einmalige Durchlaufen aller Zustandsübergänge, das z. B. zu folgenden Testfällen führt:
  - Cut, Offhook => Pending, Onhook => Cut
  - Cut, Offhook => Pending, Timeout => Timeout occured, Onhook => Cut
  - Cut, Offhook => Pending, Digit 0..9 => Checking, Rufnummer unvollständig => Pending, Digit 0..9 => Checking, ... , Rufnummer gültig => Ready, Onhook => Cut
  - Cut, Offhook => Pending, Digit 0..9 => Checking, Rufnummer unvollständig => Pending, Digit 0..9 => Checking, ... , Rufnummer ungültig => Invalid number, Onhook = Cut
- Darüber hinaus ist es sinnvoll alle Ereignisse (Events) zu testen, falls Zustandsübergäge durch mehrere Events ausgelöst werden können. Dies ergibt eine Hierarchie von Testtechniken:
  Alle Zustände ⊆ Alle Zustandsübergänge ⊆ Alle Events
- Wichtig: Test der Fehlerbehandlung nicht vergessen!

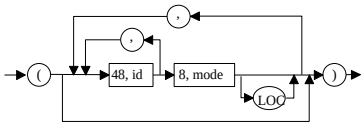
Funktionsorientierter Test
Syntaxtest

- Anwendung:

  Test von Compilerteilen mit Syntaxbeschreibungen
  - in Backus-Naur-Form bzw.
  - in Syntaxgraphen

Funktionsorientierter Test
Syntaxtest

- Beispiel: Aufbau von Parameterlisten in der Programmiersprache Chill

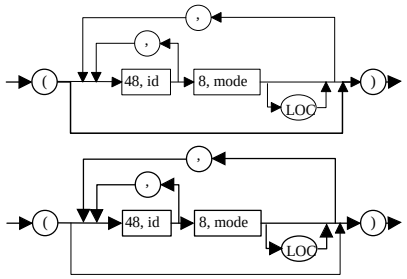


Dieser Graph kann folgendermaßen gelesen werden:
- Eine Parameterliste beginnt mit einer öffnenden runden Klammer und endet mit einer schließenden runden Klammer.
- Zwischen den Klammern stehen ein bzw. mehrere durch Kommata getrennte Bezeichener (id) gefolgt von einem mode und dem optionalen Schlüsselwort LOC.
- Dieser Inhalt der Klammern kann sich - durch Kommata getrennt - mehrfach wiederholen; er kann aber auch ganz fehlen.
- Die eckigen Symbole für id und mode verweisen auf weitere Syntaxgraphen.

Funktionsorientierter Test
Syntaxtest

- Testidee: Sytaxgraphen wie Kontrollflussgraphen behandeln
  - Abdeckung des Graphen mit Testfällen nach bestimmten Regeln (siehe strukturorientierte Testverfahren)
  - Systematische Testfallreduktion
  - Minimalkriterium bei der Überdeckung von Kontrollflussdiagrammen ist die Zweigüberdeckung, also der mindestens einmalige Durchlauf durch alle Pfeile in dem Diagramm. Dies kann auf Syntaxgraphen übertragen werden.
- Für das Beispiel ergeben sich folgende Testfälle:
  - Die leere Liste: ( )
  - z. B. die Liste: (id, id mode, id mode LOC)
- Diese Regel zur Testfallerzeugung muss als absolutes Minimum betrachtet werden.
- Schwachpunkt ist die unzureichende Prüfung der Wiederholungen.

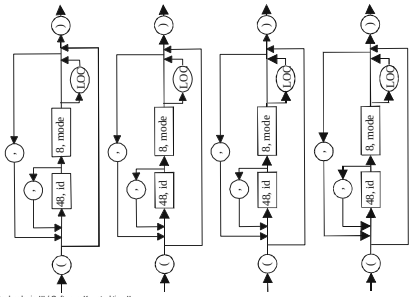
Funktionsorientierter Test
Syntaxtest



Funktionsorientierter Test
Syntaxtest

- Verbesserung:
  - Teste alle Pfade (Wege), ohne Schleifen zu durchlaufen.
  - Teste mehrere Schleifenwiederholungen, aber keine Kombinationen von Schleifenwiederholungen
- Dies führt zu folgenden Testfällen:
- Dies entspricht z. B. folgenden Parameterlisten:
  - Der leeren Liste: ( )
  - Der Liste: (id mode)
  - Der Liste: (id mode LOC)
  - z. B. der Liste: (id, id mode, id mode LOC)

Funktionsorientierter Test
Syntaxtest



Funktionsorientierter Test
Syntaxtest

- Dieses Verfahren testet Kombinationen von Zweigen des Graphen, soweit sie außerhalb von Schleifen liegen.
- Ferner werden für Schleifen die Fälle Keine Wiederholung und Wiederholung getestet.
- Durch den Verzicht auf Kombinationen bei der Prüfung von Schleifen bleibt die kombinatorische Explosion der Testfallanzahl aus.

Funktionsorientierter Test
Syntaxtest

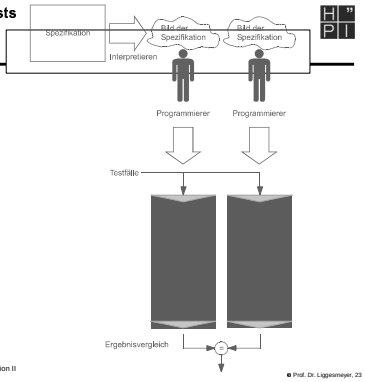
Zusätzlich: Erzeugung von Fehlerfällen

- Unkorrekte Syntax: Eine systematische Erzeugung von Testfällen ist aufgrund der Anzahl denkbarer Syntaxfehler kaum möglich. Insgesamt ist darauf zu achten, bei der Erzeugung fehlerhafter Syntax für Testfälle möglichst kleine Fehler einzufügen (z. B. Komma fehlt, Komma ist gegen Semikolon vertauscht)
- Kontextbedingungen nicht erfüllt: z. B. Prozedurname am Ende der Prozedur entspricht nicht dem Namen im Prozedurkopf, Operator passt nicht zum Datentyp des Operanden, Schlüsselwort als Bezeichner verwendet.
- Wertebereiche über- oder unterschritten: z. B.
  - Konstanten zu klein, zu groß, zu kurz oder zu lang;
  - Parameterlisten zu lang,
  - Bezeichner zu lang.

Diversifizierende Tests

- Vergleichender Test mehrerer Module.
- Die diversifizierenden Tests testen mehrere Versionen eines Moduls gegeneinander.
- Back to Back-Test:
  - Beurteilung der Ausgaben durch Vergleich der Ausgaben der unterschiedlichen Module.
  - Vorteil: Testdurchführung ist automatisierbar.
  - Nachteile: Echte Mehrfachrealisierung erforderlich. Fehler, die in allen Versionen identisch vorkommen, werden nicht erkannt.
- Mutationen-Test:
  - Eigentlich kein Testverfahren, sondern eine Möglichkeit die Leistungsfähigkeit (Fehlererkennungsquote) unterschiedlicher Testtechniken neutral zu beurteilen
  - Hoher Aufwand.

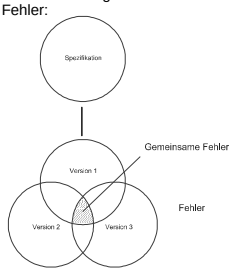
Diversifizierende Tests
Back to Back-Test: Arbeitsweise



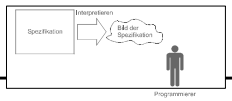
Diversifizierende Tests
Back to Back-Test

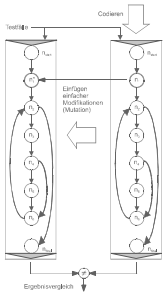
- Der Back to Back-Test verlangt die mehrfache Realisierung von Programmen, die auf einer identischen Spezifikation basieren.
- Der Back to Back-Test ist wirtschaftlich einsetzbar, falls hohe Korrektheit und hohe Zuverlässigkeit gefordert ist oder eine automatische Beurteilung der Ausgaben des Testlings gewünscht oder erforderlich ist (Echtzeitsoftware).

- Nichterkennen gemeinsamer Fehler:



Diversifizierende Tests
Mutations-Test: Arbeitsweise





Diversifizierende Tests
Mutations-Test

- Erzeugt unterschiedliche Versionen durch künstliches Einfügen von typischen Fehlern.
- Durch Variation der eingefügten Fehlertypen entstehen unterschiedliche Testverfahren.
- Es sind Testdaten zu erzeugen, die hinreichend fehlersensitiv zur Unterscheidung des Originals von der mutierten Version sind.
- Basiert auf der Erkenntnis, dass erfahrene Programmierer fast korrekte Programme erstellen. Die Programme weichen in ihrem Verhalten nur relativ geringfügig von dem vorgesehenen Verhalten ab.
- Die Differenz zwischen Ist und Soll ist im Wesentlichen die Manifestation einer Anzahl einfacher Fehler, wie Definitionen oder Referenzen falscher Variablen, unkorrekte Relationen in Prädikaten oder einfache Fehler in arithmetischen Ausdrücken.
- Da der Mutationen-Test Annahmen bezüglich der potentiellen Fehler trifft, verursacht er die Wahl von fehlersensitiven Testdaten, die gezielt die Entdeckung bestimmter Fehler ermöglichen.

Diversifizierende Tests
Mutations-Test: Mutations-Transformationen

- Referenz einer anderen Variablen
- Definition einer anderen Variablen
- Verfälschung arithmetischer Ausdrücke um multiplikative oder additive Konstanten oder Veränderung von Koeffizienten
- Änderung arithmetischer Relationen um additive Konstanten oder Verfälschung der Relation
- Verfälschung boolescher Ausdrücke

Diversifizierende Tests
Mutations-Test: Mutations-Transformationen

Beispiel: Anwendung der Mutationstransformation zur Verfälschung der relationalen Operatoren

- Mögliche relationale Operatoren: $=, \leq, \geq, <, >, \neq$ (werden in unterschiedlichen Programmiersprachen verschieden aufgeschrieben)
- Kommen in den atomaren Prädikaten eines Moduls vor.
- Zu mutierende Komponenten C der Operation ZaehleZchn :
  1. Zchn >= 'A'
  2. Zchn <= 'Z'
  3. Gesamtzahl < INT_MAX
  4. Zchn == 'A'
  5. Zchn = 'E'
  6. Zchn = 'I'
  7. Zchn = 'O'
  8. Zchn = 'U'
- Die mutierten Komponenten C' entstehen durch Vertauschung des relationalen Operators.

Diversifizierende Tests
Mutations-Test: Mutations-Transformationen

- Aus der Komponente 1 können die fünf unterschiedlichen mutierten Komponenten 1_1 bis 1_5 erzeugt werden. Die rechte Spalte gibt für jede mutierte Komponente einen Variablenwert an, der geeignet ist, die mutierte Komponente von dem Original zu unterscheiden.

1. Zchn >= 'A'

mutierte Komponente	Testdatum
1_1.Zchn == 'A'	'B'
1_2.Zchn <= 'A'	'B'
1_3.Zchn < 'A'	'A'
1_4.Zchn > 'A'	'A'
1_5.Zchn != 'A'	'A'

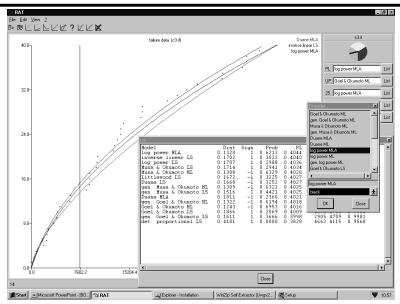
Diversifizierende Tests
Mutations-Test: Mutations-Transformationen

- Die mutierten Komponenten erzeugen mit den angegebenen Werten für *Zchn* von dem Original abweichende Ergebnisse.
  - Die Relation Zchn == 'A' ergibt mit dem Wert 'B' für *Zchn* den Wahrheitswert falsch.
  - Die unverfälschte Relation Zchn >= 'A' ist für *Zchn* gleich 'B' wahr.
  - Ein Testfall, der für Zchn den Wert 'B' enthält, ist hinreichend sensitiv zur Unterscheidung der mutierten Komponente 1_1 von der unverfälschten Komponente.
  - Mit dem Werten 'A' und 'B' für *Zchn* können alle Mutanten der Komponente 1 erkannt werden.
- Für die übrigen arithmetischen Relationen sind entsprechend geeignete Werte zu identifizieren. Die Gesamtheit dieser Werte bildet einen Satz von Testdaten, die bezogen auf die arithmetischen Relationen dieser Operation fehlersensitiv sind.

Sonstige dynamische Testtechniken

- Grenzwertanalyse
  - Die Grenzwertanalyse wählt Testdaten von den Bereichsgrenzen.
- Test spezieller Werte
  - Der Test spezieller Werte wählt diskrete Testwerte, die Sonderfälle darstellen.
- Stochastischer Test (Zufallstest)
  - Der Zufallstest wählt Testdaten zufallsgesteuert. Er ist nicht identisch mit der ad hoc-Vorgehensweise des unsystematischen Tests.
  - Da Testdaten automatisch erzeugt werden, ist er adäquat, falls eine Möglichkeit zur automatischen Beurteilung der Ergebnisse existiert und eine hohe Anzahl von Testfällen erforderlich ist (Test von Echtzeitsoftware: z. B. gemeinsam mit dem Back to Back-Test).

Auswertung eines stochastischen Tests



Dynamischer Test
Literatur

- Liggesmeyer P., Modultest und Modulverifikation - State of the Art, Mannheim, Wien, Zürich: BI Wissenschaftsverlag 1990
- Riedemann E.-H., Testmethoden für sequentielle und nebenläufige Software-Systeme, Stuttgart: Teubner 1997