Universität Stuttgart

Advanced Software Engineering

Übungsblatt 3

Maximilian Peresunchak (st152466@stud.uni-stuttgart.de)
Nico Reng (st188620@stud.uni-stuttgart.de)
Viorel Tsigos (st188085@stud.uni-stuttgart.de)
Philip Reimann (st182312@stud.uni-stuttgart.de)
Christian Keller (st166512@stud.uni-stuttgart.de)
Johannes Heugel (st183360@stud.uni-stuttgart.de)
Benedikt Wachmer (st177118@stud.uni-stuttgart.de)
Miles Holl (st180549@stud.uni-stuttgart.de)

Wintersemester 2025

30. November 2025

Aufgabe 1: Funktionsorientierter Test – Online-Ticketsystem

a)

Äquivalenzklasse	**Status**	**Beschreibung**
$ÄK_1$	gültig	Status == GEPLANT
$ÄK_2$	ungültig	Status != GEPLANT (status == ABGESAGT)
$ÄK_3$	gültig	$0 \leq$ freie Plaetze ≤ 500
$ÄK_4$	ungültig	freie Plaetze < 0
$ÄK_5$	ungültig	freie Plaetze > 500
$ÄK_6$	gültig	$1 \leq$ Anzahl Tickets ≤ 6
$ÄK_7$	ungültig	Anzahl Tickets < 1
$ÄK_8$	ungültig	Anzahl Tickets > 6
$ÄK_9$	gültig	freie Plaetze $\geq$ Anzahl Tickets
$ÄK_{10}$	ungültig	freie Plaetze $<$ Anzahl Tickets

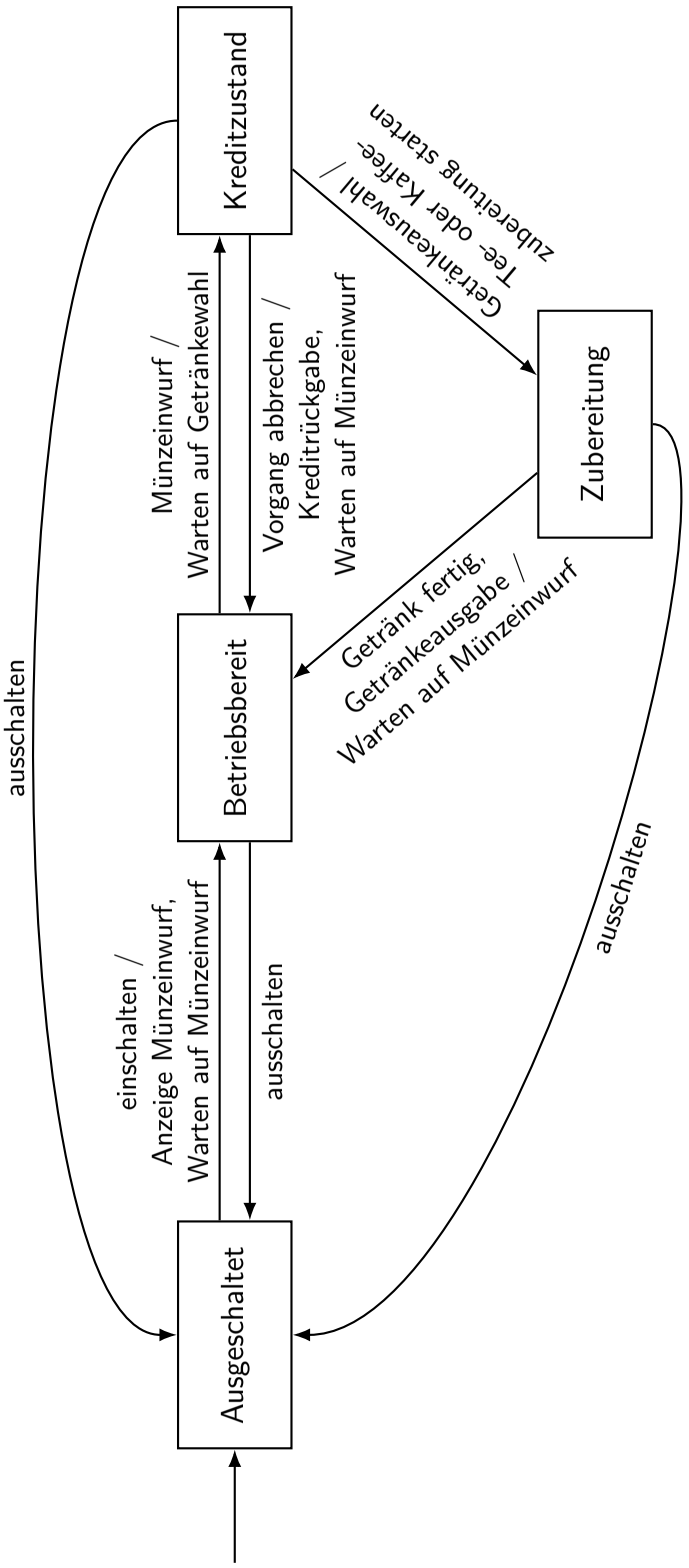
Die Teilaufgabe b) befindet sich auf der nächsten Seite.

b)

Testklasse	**Status**	**freiePlaetze**	**anzahlTickets**	**Erwartetes Ergebnis**	**Beschreibung Testcase**	**ÄK**
TK_1	GEPLANT	1	1	true	Min. Tickets, Min. Plätze	1, 3, 6, 9
TK_2	GEPLANT	500	6	true	Max. Plätze, Max. Tickets	1, 3, 6, 9
TK_3	GEPLANT	6	6	true	Max. Tickets, passend	1, 3, 6, 9
TK_4	ABGESAGT	100	2	false	Status falsch	2, 3, 6, 9
TK_5	GEPLANT	-1	1	false	Plätze edgecase min.	1, 4, 6, 10
TK_6	GEPLANT	-50	5	false	Plätze unter min.	1, 4, 6, 10
TK_7	GEPLANT	501	3	false	Plätze edgecase max.	1, 5, 6, 9
TK_8	GEPLANT	600	4	false	Plätze über max.	1, 5, 6, 9
TK_9	GEPLANT	400	0	false	anzahlTickets edgecase min.	1, 3, 7, 9
TK_{10}	GEPLANT	400	-25	false	anzahlTickets unter min.	1, 3, 7, 9
TK_{11}	GEPLANT	400	7	false	anzahlTickets edgecase max.	1, 3, 8, 9
TK_{12}	GEPLANT	4	5	false	freiePlaetze < anzahlTickets	1, 3, 6, 10

Aufgabe 2: Zustandsbasierter Test – Heißgetränkeautomat

a)



b)

Zustand / Ereignis	Ausgeschaltet	Betriebsbereit	Kreditzustand	Zubereitung
einschalten	*Betriebsbereit* Anzeige Münzeinwurf, warte auf Münzeinwurf	*Betriebsbereit*	*Kreditzustand*	*Zubereitung*
Münzeinwurf	*Ausgeschaltet*	*Kreditzustand* Warten auf Getränkeauswahl	*Kreditzustand*	*Zubereitung*
Getränkeauswahl	*Ausgeschaltet*	*Betriebsbereit*	*Zubereitung* Tee- oder Kaffeezubereitung starten	Zubereitung
Getränk fertig und Getränkeausgabe	*Ausgeschaltet*	*Betriebsbereit*	*Kreditzustand*	*Betriebsbereit* Warten auf Münzeinwurf
Vorgang abbrechen	*Ausgeschaltet*	*Betriebsbereit*	*Betriebsbereit* Kreditrückgabe, Warten auf Münzeinwurf	*Zubereitung*
ausschalten	*Ausgeschaltet*	*Ausgeschaltet*	*Ausgeschaltet*	*Ausgeschaltet*

c) Um eine vollständige Zustandsüberdeckung zu erreichen, müssen alle Zustände mindestens einmal besucht werden. In diesem Fall lassen sich innerhalb eines Durchlaufs alle Zustände besuchen, also reicht ein einziger Testfall aus, um alle Zustände abzudecken. Dieser sieht wie folgt aus:

- Ausgeschaltet $\xrightarrow{\text{einschalten}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{Münzeinwurf}}$ Kreditzustand
- Kreditzustand $\xrightarrow{\text{Getränkeauswahl}}$ Zubereitung
- Zubereitung $\xrightarrow{\text{ausschalten}}$ Ausgeschaltet

So werden die Zustände Ausgeschaltet, Betriebsbereit, Kreditzustand und Zubereitung jeweils einmal besucht und eine vollständige Zustandsüberdeckung erreicht. Der Test ist erfolgreich, wenn alle Zustände erreicht werden können, andernfalls schlägt er fehl.

d) Um eine vollständige Zustandsübergangsüberdeckung zu erreichen, müssen alle möglichen Übergänge zwischen den Zuständen mindestens einmal durchlaufen werden. In diesem Fall gibt es insgesamt 8 Übergänge, die abgedeckt werden müssen. Drei mögliche Testfälle, die zusammen alle Übergänge abdecken, sehen wie folgt aus:

i.
- Ausgeschaltet $\xrightarrow{\text{einschalten}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{Münzeinwurf}}$ Kreditzustand
- Kreditzustand $\xrightarrow{\text{Getränkeauswahl}}$ Zubereitung
- Zubereitung $\xrightarrow{\text{Getränk fertig, Getränkeausgabe}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{Münzeinwurf}}$ Kreditzustand
- Kreditzustand $\xrightarrow{\text{Vorgang abbrechen}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{ausschalten}}$ Ausgeschaltet

ii.
- Ausgeschaltet $\xrightarrow{\text{einschalten}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{Münzeinwurf}}$ Kreditzustand
- Kreditzustand $\xrightarrow{\text{ausschalten}}$ Ausgeschaltet

iii.
- Ausgeschaltet $\xrightarrow{\text{einschalten}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{ausschalten}}$ Ausgeschaltet
- Ausgeschaltet $\xrightarrow{\text{einschalten}}$ Betriebsbereit
- Betriebsbereit $\xrightarrow{\text{Münzeinwurf}}$ Kreditzustand
- Kreditzustand $\xrightarrow{\text{Getränkeauswahl}}$ Zubereitung
- Zubereitung $\xrightarrow{\text{ausschalten}}$ Ausgeschaltet

Das Testszenario ist erfolgreich, wenn alle Übergänge mindestens einmal durchlaufen wurden, andernfalls schlägt es fehl. Sprich, wenn ein Testfall fehlschlägt, schlägt das gesamte Szenario fehl.

e) Um eine vollständige Ereignisüberdeckung zu erreichen, müssen alle möglichen Ereignisse mindestens einmal ausgelöst werden. Um dies zu automatisieren, nutzen wir eine modifizierte Tiefensuche. Dazu nehmen wir uns eine Datenstruktur zur Hilfe, welche alle bereits ausgelösten Ereignisse speichert.

Die Tiefensuche wird so modifiziert, dass sie bei der Verfolgung eines Pfades nur Zweige in Betracht zieht, die ein bisher noch nicht ausgelöstes Ereignis enthalten. Sobald ein Pfad endet, weil alle möglichen Übergänge zu bereits ausgelösten Ereignissen führen, müsste die Suche zurückgehen und einen anderen Pfad verfolgen. Da dies jedoch nicht so einfach möglich ist, wird stattdessen ein neuer Testfall gestartet, der am Anfangszustand beginnt und erneut eine Tiefensuche durchführt, dieses Mal aber den zunächst zu betrachtenden Übergang auf seinem Pfad wählt.

Dies wird so lange wiederholt, bis alle Ereignisse mindestens einmal ausgelöst wurden. Dieser Algorithmus funktioniert allerdings nur für zusammenhängende Zustandsautomaten, bei denen gewährleistet ist, dass sich alle Ereignisse von einem gemeinsamen Startzustand aus erreichen lassen. Sollte es mehrere Startzustände geben, würde der Algorithmus für jeden Startzustand separat ausgeführt werden müssen, was aber zu keinem Problem führt.

Das Testszenario ist erfolgreich, wenn die Datenstruktur am Ende alle Ereignisse enthält, andernfalls schlägt es fehl.

Aufgabe 3: Kontrollflussorientierter Test – Rabattberechnung

a)

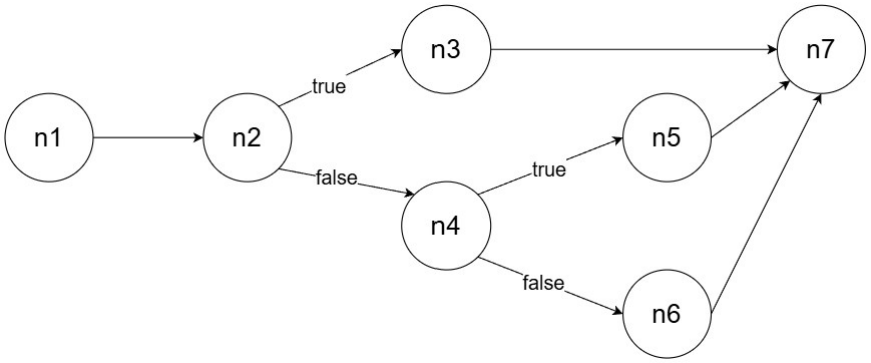


Abbildung 1: Kontrollflussgraph

n1: Der Methodeneintritt (kann auch n_{start} genannt werden)
n2: if (alter $<$ 18 || student)
n3: Return true für den ersten Fall (ersten If-Block)
n4: (mitglied && alter $<$ 65)
n5: Else if und return true
n6: Return false für die gesamte Eingabe
n7: Austritt der Methode (kann auch n_{final} genannt werden)

b)
- Pfad 1 ((alter $<$ 18 || student) ist true):
  n1 $\rightarrow$ n2 $\rightarrow$ n3 $\rightarrow$ n7
- Pfad 2 ((alter $<$ 18 || student) ist false und (mitglied && alter $<$ 65) ist true):
  n1 $\rightarrow$ n2 $\rightarrow$ n4 $\rightarrow$ n5 $\rightarrow$ n7
- Pfad 3 (Beide If-Blöcke sind false):
  n1 $\rightarrow$ n2 $\rightarrow$ n4 $\rightarrow$ n6 $\rightarrow$ n7

c) Testfall 1:
Eingabe: alter 17, mitglied false, student true. $\Rightarrow$ Pfad 1
Ausgabe: true

Testfall 2:
Eingabe: alter 25, mitglied true, student false. $\Rightarrow$ Pfad 2
Ausgabe: true

Testfall 3:
Eingabe: alter 20, mitglied false, student false $\Rightarrow$ Pfad 3
Ausgabe: false

Anweisungsüberdeckung und Zweigüberdeckung werden bei dieser Methode über die selbe minimale Menge an Testfällen erreicht.

d) Die aufgelisteten Pfade aus Teilaufgabe b) können alle theoretisch ausgeführt werden. Eine jedoch mögliche Eingabe wie beispielsweise: alter = 17, student = true und mitglied = true, wird im 1. if-Block (alter $<$ 18 || student) direkt akzeptiert und könnte nicht vom 2. Statement (mitglied && alter $<$ 65) akzeptiert werden, da dieser Block im Code nie betreten wird, da wie gesagt die erste if-Anweisung direkt akzeptiert. Somit würde hier der 1. Pfad akzeptieren und der 2. Pfad (der auch akzeptieren könnte) nicht.