Generating Test Cases from semi-formal Use Case Descriptions

Testfallgenerierung aus semi-formalen Use Cases

Lars Borner, Timea Illes-Seifert, Barbara Paech
Universität Heidelberg, Im Neuenheimer Feld 326, 69120
{borner, illes, paech}@informatik.uni-heidelberg.de

Abstract

Zahlreiche wissenschaftliche Arbeiten thematisieren die Ableitung von Testfällen aus Anforderungen, insbesondere aus Use Cases. Die entwickelten Methoden werden jedoch nur selten in Anforderungsmanagement- oder Testwerkzeugen integriert. In dieser Arbeit stellen wir einen leichtgewichtigen semi-formalen Ansatz vor, der die automatische Ableitung von logischen Testfällen aus Use Cases ermöglicht und leicht in bestehende Anforderungsmanagementwerkzeuge integriert werden kann. Hierbei werden logische Testfälle aus Use Cases unter Berücksichtigung des Kontrollflusses und der vom Use Case verarbeiteten Daten und ihrer Bedingungen abgeleitet.

Abstract

A multitude of research work has been published focussing on the derivation of test cases from requirements, particularly from use cases. However, these approaches are rarely integrated into requirements management respectively in test management tools. In this paper, we propose a lightweight, semiformal approach which allows test cases to be automatically generated from structured use case descriptions and which can easily be integrated into existing requirements management tools. The approach generates test cases by considering the control flow and the data and its constraints specified in use cases.

1 Introduction

Use Cases (UC) haben sich als Mittel zur Anforderungsermittlung und –spezifikation im objektorientierten Umfeld bewährt [Co00]. Zahlreiche wissenschaftliche Ansätze zur Ableitung von Testfällen aus UCs wurden bereits vorgeschlagen, ein Überblick vorhandener Ansätze findet sich in [IP06a]. Es existieren zahlreiche Requirements Engineering- [Ho05] und Testwerkzeuge [Il06] mit ihren jeweiligen Stärken im „eigenen“ Anwendungsbereich. Dennoch gibt es wenige Werkzeuge, die einen solchen Übergang unterstützen. Die Verknüpfung zwischen Requirements Engineering- (RE) und Testwerkzeugen beschränkt sich jedoch meistens auf die Nachvollziehbarkeit zwischen den Anforderungen und den Testfällen.

RE Werkzeuge sind für die Unterstützung der Arbeit der Anforderungsingenieure konzipiert. Sie erleichtern die Spezifikation der Anforderungen in Form von UCs und dienen somit vor allem als

Kommunikationsmedium zwischen Anforderungsingenieuren und Kunden. Aus diesem Grund werden häufig Textfelder zur Spezifikation der UC Inhalte verwendet. Hieraus resultieren Probleme bei der automatisierten Ableitung von Testfällen aus UCs. Ein Hauptproblem stellt die *geringe Strukturierung* der UCs in Werkzeugen dar. Komplexe Alternativen sowie der Kontrollfluss innerhalb eines UCs können nur sehr schwer innerhalb der Textfelder spezifiziert werden. Die so erstellen UCs sind sehr fehleranfällig und schwer wartbar. Ein weiteres Problem resultiert aus der Tatsache, dass Tester als „Stakeholder" der UC Spezifikation nicht berücksichtigt werden. Somit fehlen in den UCs *spezielle Informationen*, die für den Test notwendig sind. Insbesondere können Schritte und Daten häufig nicht getrennt spezifiziert werden. Aufgrund der bisher aufgezählten Probleme, ist die automatische Ableitung von Testfällen aus der UC-Spezifikation nicht möglich. Somit müssen die Testfälle manuell spezifiziert werden, wodurch eine hohe Redundanz zwischen den UC Beschreibungen und Testfallspezifikationen entsteht, vor allem was die auszuführenden Schritte betrifft.

Am Lehrstuhl für Software Engineering der Universität Heidelberg wird das Software Engineering Werkzeug Sysiphus [Sys07] für Forschungs- und Lehrzwecke eingesetzt. Es bietet unter anderem die Möglichkeit, Anforderungen (in Form von UCs) und Testfälle in vordefinierten Vorlagen zu spezifizieren. Jedoch fehlt auch in Sysiphus der werkzeuggestützte Übergang von UCs zu Testfällen. In dieser Arbeit stellen wir einen leichtgewichtigen Ansatz zur automatischen Ableitung von logischen Testfällen aus semi-formalen UCs vor. In einem ersten Schritt identifizieren wir Anforderungen an ein UC Modell, das die Testfallgenerierung aus nicht formalen UCs ermöglicht (Kapitel 2) und stellen dieses UC Modell vor (Kapitel 3). Anschließend detaillieren wir den Ansatz zur Testfallgenerierung (Kapitel 4). Hierbei stellen wird die Ansätze zur Generierung von Testfällen mit Hilfe der kontrollflussbasierten und der kombinatorischen Testtechnik vor. Abschließend stellen wir verwandte Arbeiten vor (Kapitel 5), fassen die Ergebnisse zusammen und geben einen Ausblick auf weitere Arbeiten (Kapitel 6), die sich vor allem auf die Kombination der beiden Testtechniken fokussieren werden.

2 Anforderungen an ein UC Modell

Basierend auf den oben beschriebenen Problemen, ergeben sich die nachfolgenden Anforderungen an ein gemeinsames UC Modell für den Anforderungsingenieur und den Tester, welches die automatische Ableitung von Testfällen unterstützt. Folglich sollte das UC Modell einerseits leichtgewichtig sein und die natürlichsprachige Spezifikation beibehalten (**A1**), um als Kommunikationsmedium zwischen Anforderungsingenieuren und Kunden verwendet werden zu können. Andererseits sollte es für Testzwecke geeignet sein. Dies schließt eine bessere Strukturierung der UC Spezifikation (**A2**), insbesondere die Trennung von Daten und Schritten (**A2.1**); eine explizite Darstellung von Alternativen (**A2.2**) sowie deren Bedingungen (**A2.3**) ein.

3 UC Modell

Das von uns entwickelte UC-Modell erweitert das in [BD03] vorgestellte UC-Modell bestehend aus Vorbedingungen, Aktorschritten, Systemschritten, Nachbedingungen sowie Ausnahmen, um die oben identifizierten Anforderungen zu erfüllen. Das entstandene Modell ist in Abbildung 1 dargestellt. Dabei stellt Abbildung 1a) eine vereinfachte Sichtweise auf das Modell dar. Ein UC beginnt hierbei mit der Vorbedingung. Dieser Vorbedingung folgt genau ein Schritt. Dieser Schritt beinhaltet eine Reihe (1..*) von Auswahlen. Die Auswahlen ermöglichen die Spezifikation von komplexen Kontrollflüssen innerhalb des UCs. Innerhalb einer Auswahl ist festgelegt, ob ihr ein weiterer Schritt, die Nachbedingung oder eine Ausnahme folgt. Eine Ausnahme beschreibt hierbei einen Abbruch des UCs auf Wunsch des Aktors bzw. aufgrund von Bedingungen. Das Erreichen einer Ausnahme bedeutet, dass die Nachbedingung des UCs unter keinen Umständen mehr erreicht werden kann (es sei denn durch nochmaliges Starten des UCs).

In Abbildung 1b) wird das einfache UC-Modell erweitert. Ein Schritt aus a) kann hierbei ein Aktor- oder Systemschritt sein. Ein *Aktorschritt* umfasst alle Handlungsalternativen, die einem Aktor[1] zu einem entsprechenden Punkt im UC zur Verfügung stehen. Diese Alternativen sind in Form von *Aktorauswahlen* spezifiziert ***(A2.2)***. Innerhalb einer Aktorauswahl kann der Aktor verschiedene Daten (0..*) eingeben ***(A2.1)*** und/oder verschiedene Aktionen ausführen. Diese Aktionen werden in textueller Form innerhalb des jeweiligen Aktorschritts definiert ***(A1)***. Weiterhin definiert jede Aktorauswahl, ob ihr ein Schritt (Aktor-, Systemschritt) oder die Nachbedingung bzw. eine Ausnahme folgt.

Ein *Systemschritt* beschreibt, wie das System auf die Aktion des Aktors reagiert. Das Verhalten wird durch die im Systemschritt enthaltenen *Systemauswahlen* definiert. Ein Systemschritt enthält, ähnlich dem Aktorschritt, eine Menge von Systemauswahlen (1..*) ***(A2.2)***. Innerhalb einer Systemauswahl können Bedingungen definiert werden. Eine Bedingung kann sich auf ein Datum beziehen und dieses Datum einschränken (z.B. Datum: *Leser.ID*, Bedingung: *ist gültig*) oder eine allgemeingültige Bedingung sein, welche sich nicht auf Daten im System bezieht. Bedingungen können durch boolesche Operatoren miteinander verknüpft werden ***(A.2.3)***. Nur wenn das Ergebnis dieser Operation „TRUE“ ergibt, wird der dazugehörige „Nachfolgeschritt“ ausgeführt (entweder ein weiterer System- oder ein Aktorschritt) oder das Ende des UCs (durch ein „reguläres“ Ende oder durch das Auftreten eines Ausnahmenfalls) erreicht. Das Erreichen eines „regulären“ Endes bedeutet, dass die im UC definierte *Nachbedingung* gilt. Vor- und Nachbedingungen beinhalten ebenfalls Bedingungen, welche beschreiben, welche Bedingungen vor bzw. nach der erfolgreichen Ausführung (d.h. durch das Erreichen des „regulären“ Endes) gelten müssen. Dabei können diese Bedingungen beliebig komplex sein und durch logische Operatoren kombiniert werden.

---

[1] Ein Aktor ist ein/e potenzielle/r BenutzerIn der Software, der/die den betrachtetet Use Case ausführt.

Daten werden in unserem Modell als eigene Entitäten modelliert, wobei ein Datum mehrere Attribute besitzen kann.

Mit Hilfe unserer UC-Beschreibung können komplexe Kontrollflüsse spezifiziert werden. Alle Inhalte der UC Modellelemente werden dabei weiterhin natürlichsprachig verfasst ***(A1)***, wodurch sie für den Anforderungsingenieur, den Kunden und den Testern lesbar und verständlich sind.

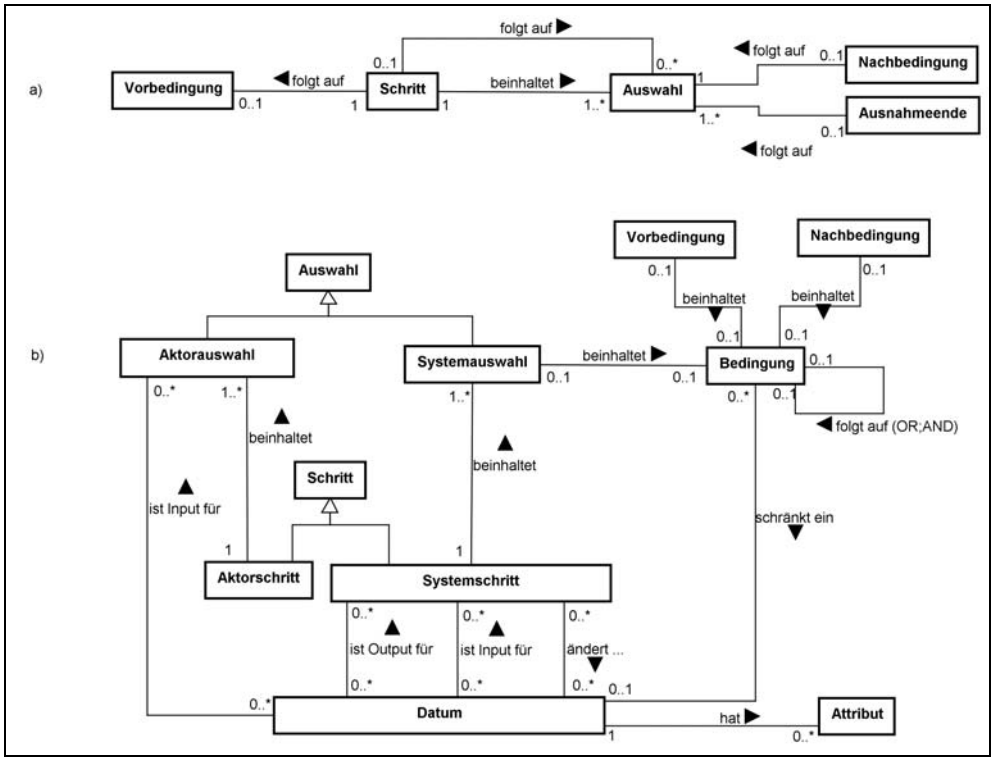


Abbildung 1: Neues Metamodell der UC Spezifikation in Sysiphus

Das Modellierungskonzept soll anhand eines Beispiels erläutert werden. Der verwendete Beispiels-UC beschreibt eine funktionale Anforderung an ein Bibliothekssystem, dass die Bibliothekarin bei der täglichen Arbeit unterstützen soll. Der UC spezifiziert den Ablauf der Kommunikation zwischen der Bibliothekarin und dem System bei der Ausleihe eines (oder mehrerer Bücher). In einem ersten Schritt initiiert sie den UC, worauf sie anschließend die Leser-ID des Lesers, der das Buch ausleihen möchte in das System eingibt. Anschließend überprüft das System ob der Leser berechtigt ist, Bücher auszuleihen, oder ob er/sie bereits zu hohe Mahngebühren zu zahlen oder bereits fünf Bücher ausgeliehen hat. Für den Fall, dass der Leser berechtigt ist, Bücher auszuleihen, gibt die Bibliothekarin die Buch-ID des auszuleihenden Buches ein. Daraufhin prüft das System, ob das Buch ausgeliehen werden kann (d.h. das Buch ist kein Referenzbestand und auch nicht vorgemerkt). Für den Fall, dass das Buch ausgeliehen werden kann wird das Buch vom System auf dem Konto des Lesers verbucht. Anschließend hat die Bibliothekarin die Möglichkeit, eine weitere Buch-ID einzugeben, oder den UC zu beenden.

Ein Beispiel dieses UCs modelliert mit unserem UC Modells in XML Repräsentation ist im Anhang 1 dargestellt[2]. Hierbei werden alle möglichen Alternativen (die aufgrund der Übersichtlichkeit in der oben dargestellten Textform weggelassen wurden) ergänzt. Das Beispiel beinhaltet eine Vorbedingung, eine reguläre Nachbedingung, eine Ausnahme (*Leserkonto erlaubt keine Ausleihe*), vier Aktorschritte und sieben Systemschritte. Die vier Aktorschritte enthalten zusammen fünf Aktorauswahlen und die sieben Systemschritte beinhalten zehn Systemauswahlen. Der UC verwendet zwei Datenentitäten (*Leser, Buch*), welche sechs Attribute beinhalten, deren Zustände das Verhalten des Systems definieren.

Im Nachfolgenden Kapitel wird erläutert, wie basierend auf dieses Modell, Testfälle mit Hilfe verschiedener Testtechniken abgeleitet werden können.

4 Testfallgenerierung

Die Elemente des UC Modells lassen sich auf die Elemente eines Testmodells abbilden. Im von uns definierten Testmodell besteht ein Testfall aus *Vorbedingungen*, einer Folge von *Test*-- und *Prüfschritten* sowie aus *Nachbedingungen*. Testschritte bezeichnen vom Tester auszuführende Aktionen, einschließlich der *Eingabeparameter*. Diese Information ist in den Aktorauswahlen enthalten. Prüfschritte hingegen umfassen eine Beschreibung der erwarteten Reaktionen des Systems sowie der *Ausgabeparameter* und entsprechen den Systemschritten. Der Gültigkeitsbereich der Ein- und Ausgabeparameter kann mit Hilfe von *Äquivalenzklassen* eingegrenzt werden. Die Äquivalenzklassen entsprechen den Bedingungen im UC Modell und dienen der Beschreibung der logischen Ein- und Ausgabedaten. Die so definierbaren Testfälle sind folglich logische Testfälle [SL06]. Diese können aus dem UC Modell durch Anwendung kontrollflussbasierter sowie kombinatorischer Testtechniken automatisch abgeleitet werden. Im nachfolgenden soll die Ableitung der Testfälle mit Hilfe der kontrollflussbasierten und der kombinatorischen Testtechnik vorgestellt werden. Hierzu werden im Kapitel 4.1 die zum Verständnis der Vorgehensweisen benötigten Begriffe definiert. Im Anschluss beschreiben wir die Herleitung der Testfälle nach den unterschiedlichen Testtechniken (Kapitel 4.2 beschreibt die kontrollflussbasierte Technik, Kapitel 4.3 beschreibt die kombinatorische Technik und Kapitel **Fehler! Verweisquelle konnte nicht gefunden werden.**). Zur Veranschaulichung werden wir die Vorgehensweisen und Algorithmen anhand eines Beispiels erläutern. Als Beispiel dient uns der UC „Bücherausleihe“ (vorgestellt am Ende von Kapitel 3).

[2] Die XML-Repräsentation wurde gewählt, um die Konzepte des UC Modells zu verdeutlichen, wobei hierdurch keine Einschränkung der Repräsentation in einem Werkzeug vorgegeben werden soll.

4.1 Grundlegende Definitionen und Konzepte

Im Nachfolgenden werden wichtige Begriffe erläutert, die für das Verständnis der nachfolgend beschriebenen Testtechniken und Algorithmen notwendig sind.

Definition 1 - Kontrollflussgraph

Ein Kontrollflussgraph ist ein gerichteter Graph mit $G_k = (V, E, v_0, V^*)$

- V ist eine endliche Menge von Knoten $\{v_0, v_1, v_2, \ldots v_n, v^*\}$. Mit $v \in$ {*Vorbedingungsknoten*, *Aktorschrittknoten*, *Systemschrittknoten*, *Nachbedingungsknoten, Ausnahmeknoten*}
- $v_0 \in V$ ist der Startknoten des Graphen
- $V^* \subset V$ und repräsentiert die Menge der Endknoten. V^* - $\{v^* \in$ {*Nachbedingung*, *Ausnahmeendknoten*}$\}$
- E ist Teilmenge von V x V - $\{(v,v') \; v \in V$ und $v' \in V$ und $v \neq v^*$ mit $v^* \in V^*$ und $v' \neq v_0$). Die Elemente von E werden als *Kanten* bezeichnet.

Definition 2 - Pfad

Ein Pfad im Kontrollflussgraph ist eine Sequenz von Knoten $v_1..v_n$ mit $n \geq 2$ und $(v_i,v_{i+1}) \in E$ und $v_1 = v_0$ und $v_n = v^*$ mit $v^* \in V^*$

Definition 3 - Teilpfad

Ein Teilpfad im Kontrollflussgraph ist eine Sequenz von Knoten $v_1...v_n$ mit $n >= 1$ und $(v_i,v_{i+1}) \in E$. Mit $v_1 \neq v_0$ **oder** $v_n \neq v^*$ mit $v^* \in V^*$.

Definition 4 - Überdeckungskriterium

Gibt an, welche und wie viele Testfälle zu erstellen sind. (In Anlehnung an [SL06].

Definition 5 - Testsuite

Ist eine Menge von Testfällen, die ein gegebenes Überdeckungskriterium erfüllt.

4.2 Kontrollflussbasierte Testtechnik

Die Ableitung von Testfällen aus UCs mit Hilfe der kontrollflussbasierten Testtechnik setzt eine Transformation der UC-Beschreibung in einen Kontrollflussgraphen voraus. Hierbei wird in einem ersten Schritt für jeden UC ein entsprechender Kontrollflussgraph erstellt. Das von uns vorgeschlagene UC-Modell (vgl. Kapitel 3) enthält alle Informationen die für die automatische Ableitung des Kontrollflussgraphen aus der UC-Beschreibung benötigt werden. Algorithmus 1 beschreibt wie die Ableitung der Kontrollflussgraphen erfolgt.

Input: Use Case Beschreibung **UC**
Output: Kontrollflussgraph G_k
I. Erstelle eine leere Schrittliste (Fifo) *L*.
II. Erstelle einen Knoten v_0 für die Vorbedingung in *UC*.
III. Identifiziere den Nachfolgeschritt *s* (ein Aktorschritt) für die Vorbedingung, erstelle einen Knoten *v*, der *s* repräsentiert, trage *s* in die Schrittliste ein und verbinde v_0 mit *v*.

IV. Solange *L* nicht leer ist, tue:

1. Nimm nächsten Schritt *s* aus der Liste
2. Wenn *s* ein Aktorschritt ist, dann identifiziere alle Aktorauswahlen für *s*. Für jede Aktorauswahl in *s* tue:
   a) Identifiziere den Nachfolgeschritt *s'* (Aktorschritt, Systemschritt, Nachbedingung, Ausnahmeende)
   b) Wenn *v'* (der Knoten der *s'* in Gk repräsentiert) nicht in G_k ist, dann erstelle für *s'* einen Knoten *v'* in G_k der *s'* in $\mathbf{G_k}$ repräsentiert und trage *s'* in die Schrittliste ein.
   c) Verbinde *v'* mit *v* (wobei *v* der Knoten in G_k ist, der *s* repräsentiert), beschrifte die Kante mit dem Namen der Aktorauswahl
3. Wenn *s* ein Systemschritt ist, dann identifiziere alle Systemauswahlen für *s*. Für jede Systemauswahl in **s** tue:
   a) Identifiziere den Nachfolgeschritt *s'* (Aktorschritt, Systemschritt, Ausnahme, Nachbedingung, Ausnahmeende)
   b) Wenn *v'* (der Knoten der *s'* in G_k repräsentiert) nicht in G_k ist, dann erstelle für *s'* einen Knoten *v'* in $\mathbf{G_k}$ der *s'* in $\mathbf{G_k}$ repräsentiert und trage *s'* in die Schrittliste ein.
   c) Verbinde *v'* mit *v* (wobei *v* der Knoten in G_k ist, der *s* repräsentiert), beschrifte die Kante mit den Daten und ihren Einschränkungen sowie Bedingungen enthalten in der Systemauswahl.
4. Ist **s** eine Nachbedingung → tue nichts.
5. Ist **s** ein Ausnahmeende → tue nichts..

Algorithmus 1: Algorithmus zur Ableitung des Kontrollflussgraphs aus der UC-Beschreibung

Das Ergebnis der Anwendung des Algorithmus auf das Beispiel des UCs „Bücherausleihe“ ist in Abbildung 2 zu sehen. Auf Basis des so entstandenen Kontrollflussgraphen können Testfälle für ein vordefiniertes Überdeckungskriterium definiert werden.

Definition 6 - Testfall

Ein Testfall stellt einen möglichen Pfad durch den Kontrollflussgraphen dar.

Überdeckungskriterien für die kontrollflussbasierte Testtechnik

Für den Kontrollflussgraphen, der einen UC, abgeleitet aus einer UC Beschreibung, können die folgenden Überdeckungskriterien zur Ableitung von Testfällen definiert werden:

Definition 7 - Aktorschrittüberdeckung

Gibt an, dass jeder Aktorschritt (repräsentiert durch einen Knoten im Kontrollflussgraph) in mindestens einem Testfall der Testsuite enthalten sein muss.

Definition 8 - Systemschrittüberdeckung

Gibt an, dass jeder Systemschritt (repräsentiert durch einen Knoten im Kontrollflussgraph) in mindestens einem Testfall der Testsuite enthalten sein muss.

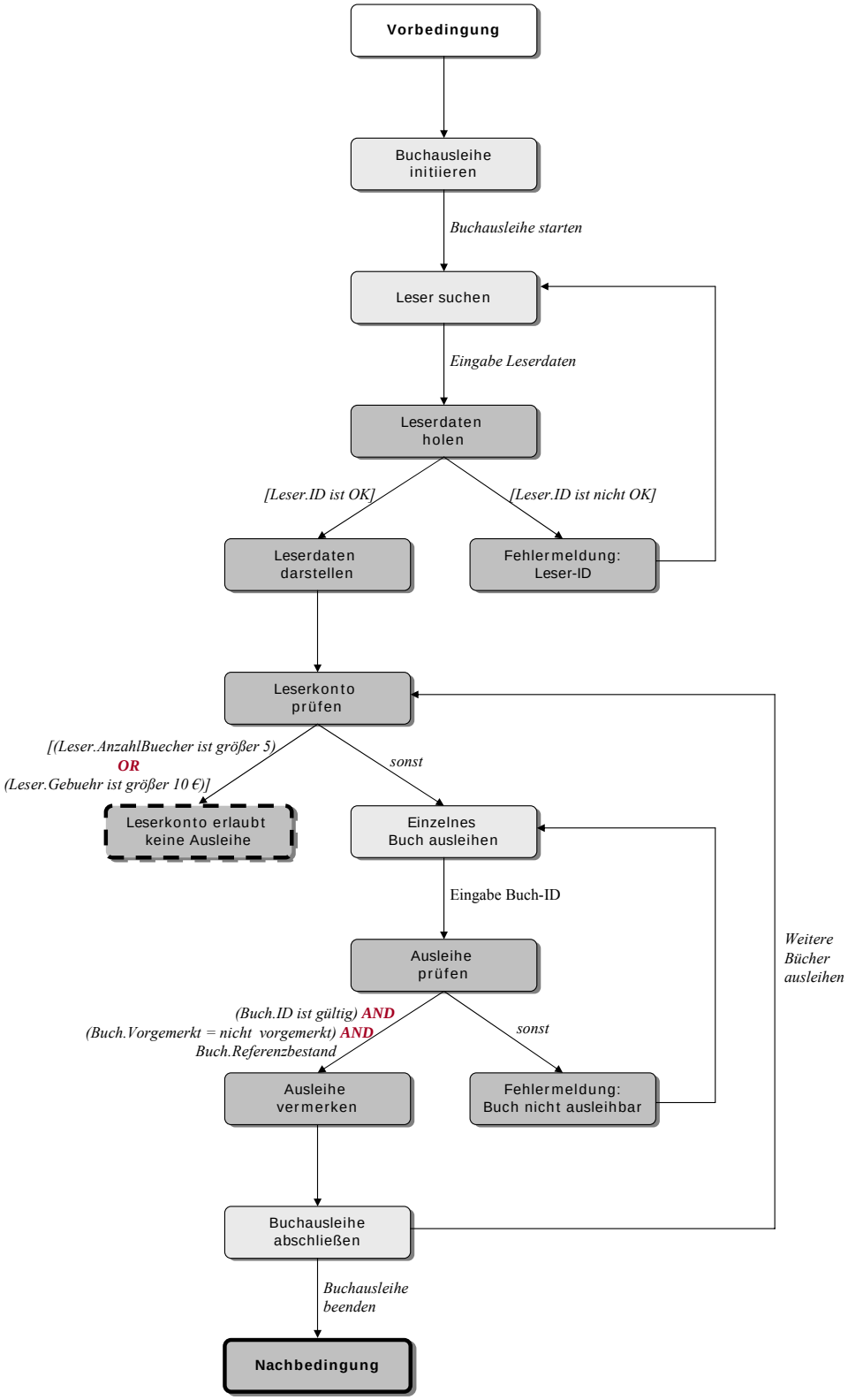


Abbildung 2: Kontrollfluss des UCs „Buchausleihe“

Definition 9 - Ausnahmeüberdeckung

Gibt an, dass jedes Ausnahmeende (repräsentiert durch einen Endknoten im Kontrollflussgrap) in mindestens einem Testfall der Testsuite enthalten sein muss.

Definition 10 - Endknotenüberdeckung

Gibt an, dass jede Ausnahmen bzw. die Nachbedingung (repräsentiert durch einen Endknoten im Kontrollflussgraph) in mindestens einem Testfall der Testsuite enthalten sein müssen.

Definition 11 - Schrittüberdeckung

Gibt an, dass jeder Schritt (Aktor- und Systemschritt, repräsentiert durch einen Knoten im Kontrollflussgraph) in mindestens einem Testfall der Testsuite enthalten sein muss.

Definition 12 - Kantenüberdeckung (in Anlehnung an die Zweigüberdeckung)

Gibt an, dass jede Kanten $\boldsymbol{e} \in \boldsymbol{E}$ in mindestens einem Testfall der Testsuite enthalten sein muss.

Ein Vergleich der verschiedenen Überdeckungskriterien illustriert Abbildung 3 dar. Abbildung 3a stellt hierbei den vollständigen Kontrollfluss eines möglichen UCs dar, wobei Aktorschritte mit **A*x*** und Systemschritte mit **S*x*** gekennzeichnet sind. Weiterhin sind Ausnahmen mit E*x* gekennzeichnet. In den Abbildungen „b“ bis „g“ wurden die Knoten und Kanten farbig markiert, die zu Testfällen führen, die ein bestimmtes Überdeckungskriterium erfüllen. Tabelle 1 stellt die resultierenden Testsuiten für die unterschiedlichen Überdeckungskriterien dar, wobei Anfangs- und Endknoten nicht aufgelistet wurden.

Überdeckung	**Testfallmenge in der resultierenden Testsuite**
b - Aktorschrittüberdeckung	{ **TF1** (*A1, A2, A4*), **TF2** (*A1, A3, S2, A4*)}
c - Systemschrittüberdeckung	{ **TF1** (*A1, A3, S2, A4*), **TF2** (*A1, S1, E1*)}
d - Ausnahmeüberdeckung	{ **TF1** (*A1, A2, E2*), **TF2** (*A1, S1, E1*)}
e - Endknotenüberdeckung	{ **TF1** (*A1, A2, E2*), **TF2** (*A1, A2, A4*), **TF3** (*A1, S1, E1*)}
f – Schrittüberdeckung	{ **TF1** (*A1, A2, E2*), **TF2** (*A1, A3, S2, A4*), **TF3** (*A1, S1, E1*)}
g - Kantenüberdeckung	{ **TF1** (*A1, A2, E2*), **TF2** (*A1, A2, A4*), **TF3** (*A1, A3, S2, A4*), **TF4** (*A1, S1, A4*), **TF5** (*A1, S1, E1*)}

Tabelle 1 – Testsuiten für unterschiedliche Überdeckungskriterien

Es ist zu erkennen, dass die Anzahl der Testfälle pro Testsuite sehr unterschiedlich ist. Während die Aktorschritt-, Systemschritt- und Ausnahmeüberdeckung sehr leicht mit jeweils zwei Testfällen pro Testsuite zu erreichen ist, werden für die Endknoten-, Schritt- und Kantenüberdeckung drei bzw. fünf Testfälle benötigt.

Das Erreichen eines vorgegebenen Überdeckungskriteriums kann durch verschiedene Mengen von Testfällen erreicht werden Die in **Fehler! Verweisquelle konnte nicht gefunden werden.** und in Tabelle 1 dargestellten Testsuiten stellen nur einen Möglichkeit dar, die gesetzten Überdeckungskriterien zu erreichen.

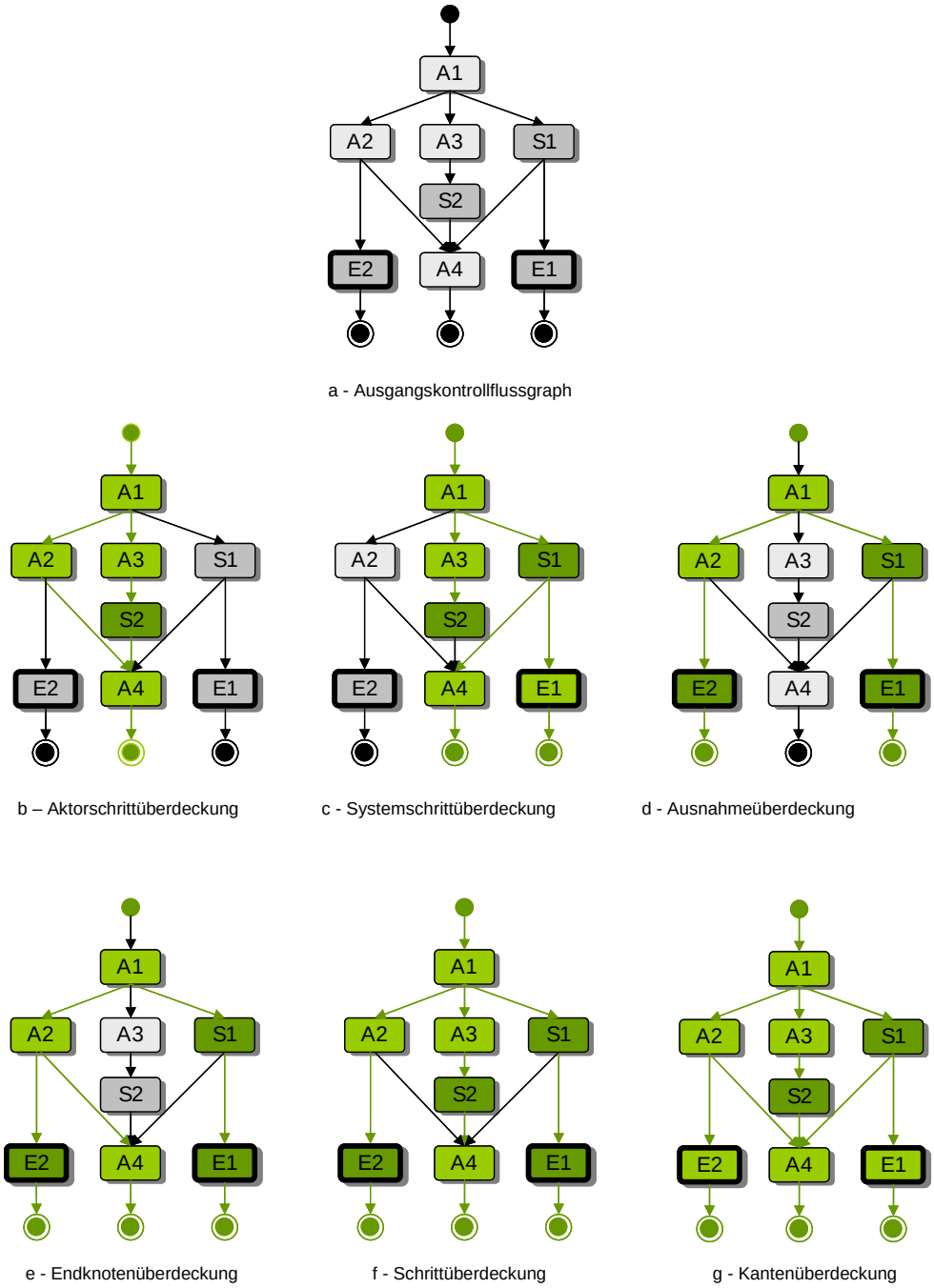


Abbildung 3: Unterschiedliche Abdeckungskriterien im Kontrollflussgraph

Durch die Vorgabe eines Überdeckungskriteriums können für den UC, repräsentiert durch den Kontrollflussgraph Testfälle abgeleitet werden. Der Algorithmus zur Ableitung der Testfälle ist in Algorithmus 2 dargestellt.

Input: Kontrollflussgraph $\mathbf{G_K}$, Überdeckungskriterium **U**
Output: Testsuite T

I. Identifiziere alle Elemente in G_k die dem Überdeckungskriterium **U** entsprechen und füge sie in die Elementemenge *M* ein.
II. Konstruiere den Kontrollflussbaum aus dem Kontrollflussgraphen[3]
III. Für jeden Pfad *p* aus der Menge P der Pfade, von der Wurzel zu den Blättern im Kontrollflussbaum, deren Blätter Element V* sind, tue:
    a. Wenn *p* Elemente aus *M* enthält, dann entferne diese Elemente aus *M* und füge *p* in *T* ein.
IV. Wenn Anzahl der Elemente in der Elementemenge größer ist als Null, dann: Für jeden Teilpfad *q* aus der Menge *Q* der Pfade, von der Wurzel zu den Blättern im Kontrollflussbaum, deren Blätter nicht Element V*, tue:
    a. Wenn *q* Elemente aus *M* enthält, dann entferne diese Elemente aus *M*.
    b. Sei Blatt *b* der Endknoten von *q,* so dass q = $(v_0,q_1q_2..q_n,b)$. Finde ein *p* in *P*, für das gilt p = $(v_0,p_1,p_2 \ldots p_k,b, p_{k+1}; \ldots p_n,v^*)$
    c. Erstelle Pfad *p'* aus q und p: p' = $(v_0, q_1 \ldots q_n,b,p_{k+1} \ldots p_n,v^*)$
    d. Füge p' in T ein.

Algorithmus 2: Algorithmus zur Ableitung des Kontrollflussgraphen aus der UC-Beschreibung

Dieser Algorithmus wurde von uns auf das UC Beispiel „Bücherausleihe“ unter Vorgabe verschiedener Überdeckungskriterien angewendet. Die Ergebnisse dieser Anwendung sind in Anhang 2 - Anhang 6 illustriert. Hierbei wurde der Kontrollflussgraph in den in Abbildung 2 dargestellten Kontrollflussbaum überführt. Die Teile des Graphen, die durch die Anwendung des jeweiligen Überdeckungskriteriums getestet werden, sind grün dargestellt, nicht ausgeführte Teile sind grau. Es ist zu erkennen, dass nur durch die Anwendung des Überdeckungskriteriums „Kantenüberdeckung“ alle Teile des Graphens mindestens einmal ausgeführt werden.

4.3 Kombinatorische Testtechnik

Die Ableitung von Testfällen nach kombinatorischen Testtechniken lehnt sich an die Black Box Testtechnik der Äquivalenzklassenbildung an. In einem ersten Schritt werden alle Daten, die im Rahmen des UCs an das System übergeben werden identifiziert. Diese sind in den Aktorauswahlen beschrieben. Anschließend werden mit Hilfe der Einschränkungen (beschrieben in den Systemauswahlen) die Äquivalenzklassen identifiziert. In einem letzten Schritt werden die Äquivalenzklassen entsprechend

[3] Der Kontrollflussbaum entsteht durch Anwendung des Algorithmus zur Ableitung eines Übergangsbaums aus einem Zustandsübergangsdiagramm im zustandsbezogenen Test (vgl. [Spillner])

bekannter Überdeckungskriterien (z.B. *each used, pair wise* oder *t-wise* [GOA05]) miteinander kombiniert, um die Testfälle abzuleiten.

Im Beispiel-UC der „Bücherausleihe“ sind die einzugebenden Daten durch die Bibliothekarin die Daten über den Leser (Leser.ID) und Daten über die auszuleihenden Bücher (Buch.ID). Die Daten und Bedingungen, die von Seiten des Systems geprüft werden, sind in Tabelle 2 dargestellt. Es ist zu erkennen, dass zu jedem gültigen Wert, seine Negation, dargestellt durch „(Nicht)“ hinzugenommen wird, diese Informationen für die Testfallerstellung wichtig ist.

Datum	**Bedingung**
Leser.ID	*Ist OK*
Leser.ID	*(Nicht) Ist OK*
Leser.AnzahlBuecher	*Ist größer 5*
Leser.AnzahlBuecher	*(Nicht)ist größer 5*
Leser.Gebuehr	*Ist größer als 10*
Leser.Gebuehr	*(Nicht)ist größer 10*
Buch.ID	*Ist gültig*
Buch.ID	*(Nicht)ist gültig*
Buch.Vorgemerkt	*Nicht vorgemerkt*
Buch.Vorgemerkt	*(Nicht)Nicht vorgemerkt*
Buch.Referenzbestand	*Nein*
Buch.Referenzbestand	*(Nicht)Nein*

Tabelle 2 – Daten und Bedingungen des Beispiel-UCs

Mit Hilfe der Informationen und einem gegebenen Überdeckungskriterium können die Testfälle abgeleitet werden. Für das Kriterium „Each used“ müssen so viele Testfälle erstellt werden, dass jede Äquivalenzklasse in mindestens einem Testfall verwendet wird. Für das Beispielsystem sind hierfür zwei Testfälle notwendig (vgl. Anhang 7). Um der paarweisen Überdeckung gerecht zu werden, müssen mindestens sechs Testfälle erstellt werden (vgl. Anhang 8).

5 Verwandte Arbeiten

Zwei Gruppen von Ansätzen zur Ableitung von Testfällen aus UCs können unterschieden werden. *Modellexplorationsansätze* belassen UCs in ihrer ursprünglichen Form und schlagen intuitive Vorgehensweisen zur Definition von Testfällen aus UCs vor. Ein Beispiel ist in [Ah02] beschrieben. Diese Ansätze bieten den Vorteil, dass nur ein Modell sowohl zur Anforderungsspezifikation als auch zur Ableitung von Testfällen erstellt und gewartet werden muss. Im Vergleich zu unserer Vorgehensweise bieten die vorgeschlagenen Ansätze aber keine für eine automatische Generierung von Testfällen ausreichende Strukturierung von UCs an.

Modelltransformationsansätze schlagen eine Vorgehensweise vor, die aus einer textuellen UC Beschreibung ein „Zwischenmodell“ ableitet, welches als Ausgangspunkt für die Generierung von Testfällen dient. Hierfür werden die Modelle nach bestimmten Überdeckungskriterien traversiert, um Testfälle abzuleiten. Als Zwischenmodelle dienen beispielsweise Zustandsautomaten [RQ03] oder Aktivitätsdiagramme [RQ03, BL02]. Andere Arbeiten schlagen eigene Modelle vor, wie Schrittgraphen [Wi99], Abhängigkeitsgraphen [RG03], mit OCL annotierte Zustandsdiagramme [Ne06] oder Zielgraphen [ARS05]. Der Vorteil der Modelltransformationsansätze ist die leichte Generierbarkeit von Testfällen. Im Vergleich zu unserem Ansatz, werden hierfür zwei unterschiedliche Modelle notwendig, die separat gewartet und weiter entwickelt werden müssen. In [IP06b] werden typische Ansätze in Form von Mustern zusammengefasst. Weitere Ansätze untersuchen den Nutzen einer frühen Ableitung von Testfällen aus UCs während der Anforderungserhebungsphase, beispielsweise in [RR98].

6 Zusammenfassung und Ausblick

In dieser Arbeit haben wir einen Ansatz zur semi-formalen Beschreibung von UCs vorgestellt, der eine automatische Ableitung von logischen Testfällen unter Berücksichtigung gegebener Überdeckungskriterien ermöglicht. Diese Testfälle können in einem anschließenden Schritt manuell mit konkreten Daten angereichert werden, um ausführbare Testfälle zu erstellen. Unser Ansatz zeigt, dass durch eine geeignete Strukturierung der UCs eine Automatisierung der Testfallableitung erreicht werden kann, ohne auf formale Sprachen zurückgreifen zu müssen. Weiterhin werden alle in Abschnitt 2 beschriebenen Anforderungen erfüllt, wobei zusätzlich das Aufdecken von Fehlern im Kontrollfluss der UCs (z.B. Sackgassen, Endlosschleifen) sowie die Unterstützung von verschiedenen Testtechniken mit unterschiedlichen Überdeckungskriterien ermöglicht werden. Der leichtgewichtige Ansatz ermöglicht keine automatische Generierung von konkreten Testdaten oder von ausführbarem Testcode, bietet aber dennoch den großen Vorteil, die Zusammenarbeit zwischen Anforderungsingenieuren und Testdesignern zu verbessern, da beide Rollen auf dem selben Modell arbeiten können. Weiterhin arbeiten wir an der Integration der kontrollflussbasierten und kombinatorischen Testtechniken, um durch die Informationen, die im Kontrollfluss enthalten sind, unmögliche Testfälle (die durch die Anwendung kombinatorischer Testtechniken entstehen können) auszuschließen und somit die Testsuite zu minimieren.

Literaturverzeichnis

[Ah02] Ahlowalia, N.: Testing from Use Cases Using Path Analysis Technique, International Conference On Software Testing Analysis & Review, 2002.

[ARS05] Alspaugh, T.A., Richardson, D.J., and Standish, T.A.: Scenarios, State Machines and Purpose Driven Testing, 4th International Workshop on Scenarios and State Machines: Models, Algorithms and Tools (SCESM'05), St. Louis, USA, (2005)

[BL02] Briand, L., Labiche, Y.: A UML-based Approach to System Testing, Technical Report, Carleton University, 2002.

[BD03] Bruegge, B.; Dutoit, A. H.: Object-Oriented Software Engineering Using UML, Patterns, and Java. Prentice Hall, Englewood Cliffs, NJ, 2003.

[Co00] Cockburn, A.: Writing Effective Use Cases, Addison-Wesley Professional, 2000

[GOA05] Grindal, M.; Offutt, J.; Andler, S.: Combination Testing Strategies: a Survey. Software Testing, Verification and Reliability, 2005; S. 167-199.

[Ho05] Hood, C. et al.: iX-Studie 01/2005 Requirements Engineering: Methoden und Techniken, Einführungsszenarien und Werkzeuge im Vergleich, Heise Zeitschriften Verlag, 2006.

[Il06] Illes, T. et al.: Software-Testmanagement Planung, Design, Durchführung und Auswertung von Tests - Methodenbericht und Analyse unterstützender Werkzeuge, Heise Zeitschriften Verlag, 2006.

[IBP07] Illes-Seifert, T., Borner, L., Paech, B.: Generating Test Cases from semi-formal use case description, Technical Report, SWEHD-TR-2007-02, 2007. http://www-swe.informatik.uni-heidelberg.de/research/publications/SWEHD_TR2007_02.pdf

[IP06a] Illes, T., Paech, B.: An Analysis of Use Case Based Testing Approaches Based on a Defect Taxonomy, In (Sacha, K.): Proceedings of the IFIP International Federation for Information Processing, Band 227, Software Engineering Techniques: Design for Quality, S. 211-222, 2006.

[IP06b] Illes, T., Paech, B.: Workshop: From "V" to "U" or: How Can We Bridge the V-Gap Between Requirements and Test?, Software & Systems Quality Conferences 2006, Düsseldorf, 2006.

[Ne06] Nebut, C. et al.: Automatic test generation: a use case driven approach, Transactions on Software Engineering, Band. 32, Ausgabe 3, 2006, S. 140-155.

[RR98] Regnell, B.; Runeson, P.: Combining Scenario-based Requirements with Static Verification and Dynamic Testing, Proceedings of 4th Intl Workshop on Requirements Engineering - Foundation for Software Quality (REFSQ'98), Pisa, Italy, Juni 1998.

[RQ03] Rupp, C., Queins, S.: Vom Use-Case zum Test-Case, OBJEKTspektrum, vol. 4, 2003.

[RG03] Ryser, J., Glinz, M.: SCENT: A Method Employing Scenarios to Systematically Derive Test Cases for System Test, Technical Report, University of Zürich, 2000/03.

[SL06] Spillner, S.; Linz, T.: Basiswissen Softwaretest, dpunkt.verlag, 2006.

[Sys07] Sysiphus, http://sysiphus.informatik.tu-muenchen.de/, 18.06.2007.

[Wi99] Winter, M.: Qualitätssicherung für objektorientierte Software - Anforderungsermittlung und Test gegen die Anforderungsspezifiktion, Dissertation, Dept. of. CS, University of Hagen, Sept. 1999.

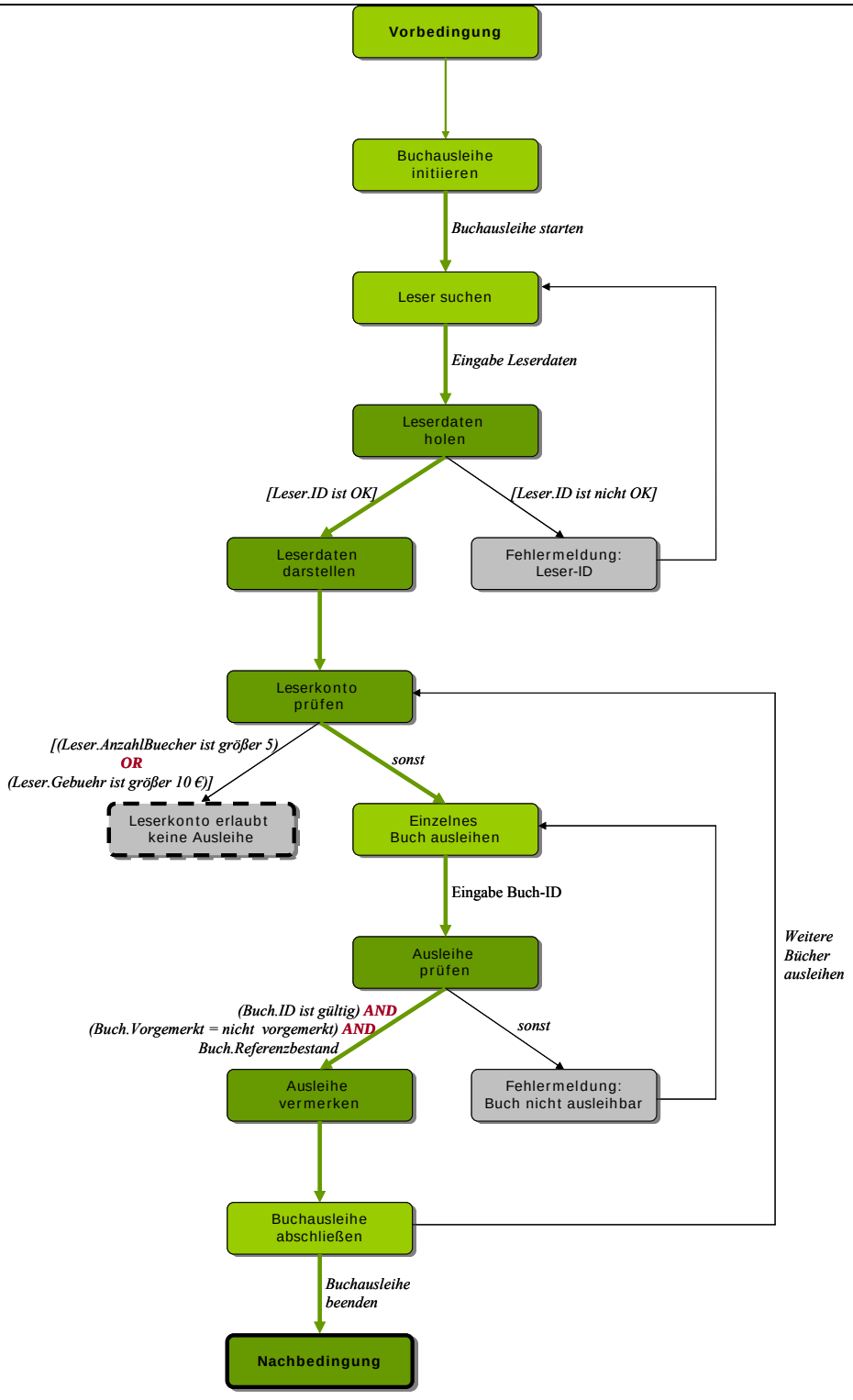
Anhang 1 - XML-Repräsentation des UC “Bücherausleihe”

```
<UseCase name="Bücherausleihe">
 <Precondition>
  <Condition>
   <ConditionText>
    Aktor ist in das System eingeloggt
   </ConditionText>
  </Condition>
  <Condition>
   <Data>Leser</Data>
   <ConditionText>
    Mindestens ein Leser ist im System vorhanden
   </ConditionText>
  </Condition>
  <ActorStep>Buchausleihe initiieren</ActorStep>
 </Precondition>

 <ActorStep name="Buchausleihe initiieren">
  <ActorChoice name = "Buchausleihe starten">
   <Description>
    Aktor teilt dem System mit, dass er/sie eine Buchausleihe vermerken will
   </Description>
   <ActorStep>Leser suchen</ActorStep>
  </ActorChoice>
 </ActorStep>

 <ActorStep name ="Leser suchen">
  <ActorChoice name = "Eingabe Leserdaten">
   <Description>
    Aktor gibt die Leserdaten ein und bestaetigt die Eingabe
   </Description>
   <Input>
    <Data>Leser.ID</Data>
   </Input>
   <SystemStep>Leserdaten holen</SystemStep>
  </ActorChoice>
 </ActorStep>

 <SystemStep name = "Leserdaten holen">
  <Description>
   System ueberprueft die eingegebenen Leserdaten
  </Description>
  <Input>
   <Data>Leser.ID</Data>
  </Input>
  <SystemChoice>
   <Condition>
    <Data>Leser.ID</Data>
    <ConditionText>ist OK</ConditionText>
   </Condition>
   <SystemStep>Daten darstellen</SystemStep>
  </SystemChoice>
  <SystemChoice>
   <Condition>
    <Data>Leser.ID</Data>
    <ConditionText>ist nicht OK</ConditionText>
   </Condition>
   <SystemStep>Fehlermeldung: Leser-ID</SystemStep>
  </SystemChoice>
 </SystemStep>

 <SystemStep name = "Fehlermeldung Leser-ID">
  <Description>
   System gibt Fehlermeldung aus
  </Description>
  <Output>
   Eingegebene Leser-ID ist unbekannt
  </Output>
  <SystemChoice>
```

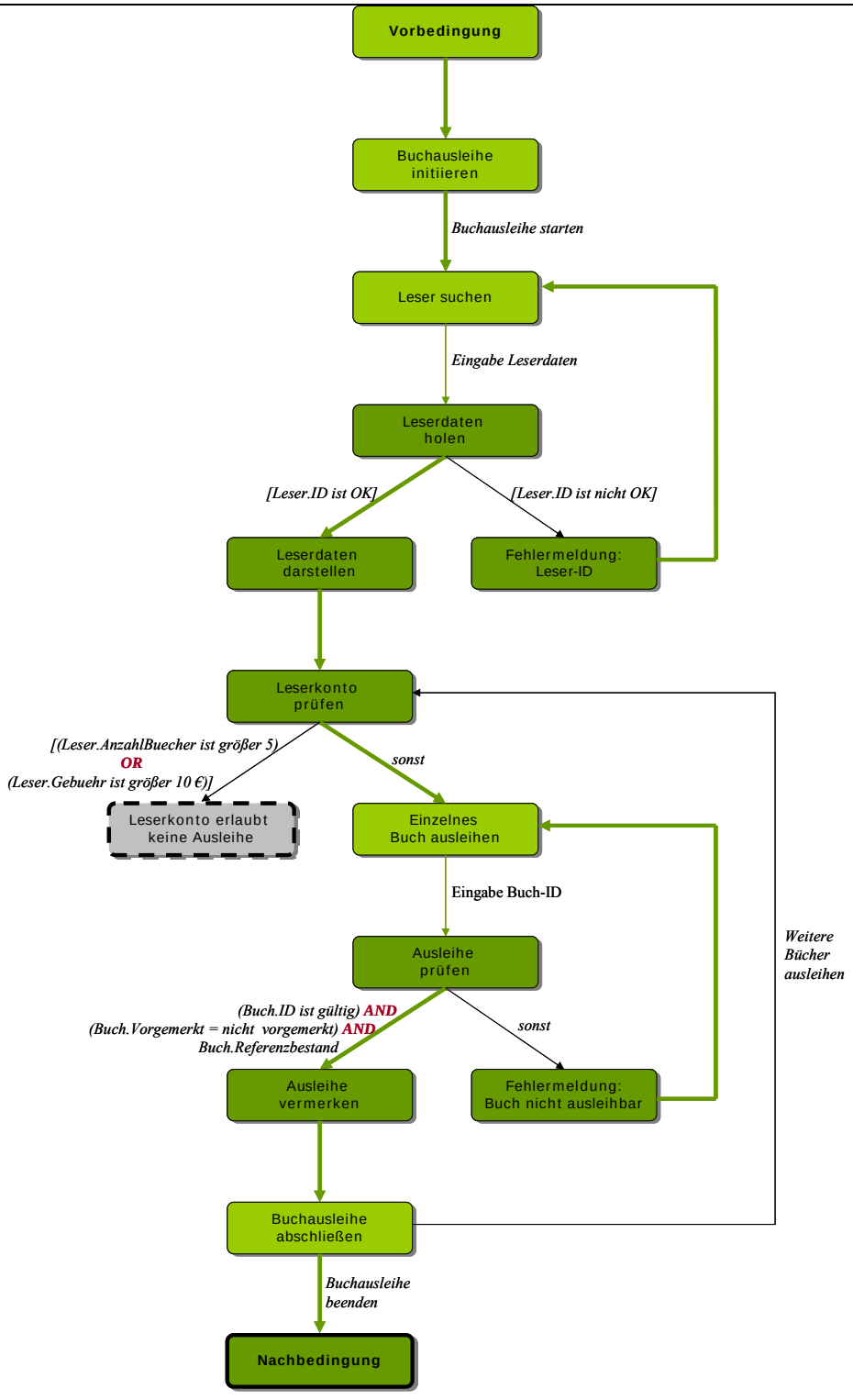
```
    <ActorStep>Leser suchen</ActorStep>
  </SystemChoice>
</SystemStep>

<SystemStep name = "Leserdaten darstellen">
  <Description>
    System holt die Leserdaten und stellt diese dar
  </Description>
  <Input>
    <Data>Leser.ID</Data>
  </Input>
  <Output>
    <Data>Leser</Data>
  </Output>
  <SystemChoice>
    <SystemStep>Leserkonto pruefen</SystemStep>
  </SystemChoice>
</SystemStep>

<SystemStep name = "Leserkonto pruefen">
  <Description>
    System ueberprueft das Leserkonto, ob Ausleihen moeglich sind
  </Description>
  <Input>
    <Data>Leser.ID</Data>
    <Data>Leser.AnzahlBuecher</Data>
    <Data>Leser.Gebuehr</Data>
  </Input>
  <SystemChoice>
    <Condition>
      <Data>Leser.AnzahlBuecher</Data>
      <ConditionText>ist groesser 5</ConditionText>
    </Condition>
    <Operator>OR</Operator>
    <Condition>
      <Data>Leser.Gebuehr</Data>
      <ConditionText>ist groesser 10 Euro</ConditionText>
    </Condition>
    <Exception>Leserkonto erlaubt keine Ausleihe</Exception>
  </SystemChoice>
  <SystemChoice>
    <ActorStep>Einzelnes Buch ausleihen</ActorStep>
  </SystemChoice>
</SystemStep>

<ActorStep name = "Einzelnes Buch ausleihen">
  <ActorChoice name = "Eingabe Buch-ID">
    <Description>
      Aktor gibt die ID des auszuleihenden Buches ein und bestätigt die Eingabe
    </Description>
    <Input>
      <Data>Buch.ID</Data>
    </Input>
    <SystemStep>Ausleihe pruefen</SystemStep>
  </ActorChoice>
</ActorStep>

<SystemStep name = "Ausleihe pruefen">
  <Description>
    System ueberprueft die eingegebene Buch-ID
  </Description>
  <Input>
    <Data>Buch.ID</Data>
  </Input>
  <SystemChoice>
    <Condition>
      <Data>Buch.ID</Data>
      <ConditionText>ist gültig</ConditionText>
    </Condition>
    <Operator>AND</Operator>
    <Condition>
```

```
      <Data>Buch.Vorgemerkt</Data>
      <ConditionText>nicht vorgemerkt</ConditionText>
    </Condition>
    <Operator>AND</Operator>
    <Condition>
      <Data>Buch.Referenz</Data>
      <ConditionText>nicht Referenzbestand</ConditionText>
    </Condition>
    <SystemStep>Ausleihe vermerken</SystemStep>
  </SystemChoice>
  <SystemChoice>
    <SystemStep>Fehlermeldung: Buch nicht ausleihbar</SystemStep>
  </SystemChoice>
</SystemStep>

<SystemStep name = "Fehlermeldung: Buch nicht ausleihbar">
  <Description>
    System gibt Fehlermeldung aus
  </Description>
  <Output>
    Buch nicht ausleihbar
  </Output>
  <SystemChoice>
    <ActorStep>Einzelnes Buch ausleihen</ActorStep>
  </SystemChoice>
</SystemStep>

<SystemStep name = "Ausleihe vermerken">
  <Description>
    Das Buch wird als ausgeliehen durch den entsprechenden Leser vermerkt
  </Description>
  <Input>
    <Data>Buch.ID</Data>
    <Data>Leser.ID</Data>
  </Input>
  <DataUpdate>
    <Data>Buch.Status</Data>
    <Description>
      ändert sich auf "ausgeliehen"
    </Description>
  </DataUpdate>
  <DataUpdate>
    <Data>Leser.Ausleihliste</Data>
    <Description>
      Der Ausleihliste des Lesers wird das neu ausgeliehene Buch hinzugefuegt
    </Description>
  </DataUpdate>
  <SystemChoice>
    <ActorStep>Buchausleihe abschliessen</ActorStep>
  </SystemChoice>
</SystemStep>

<ActorStep name = "Buchausleihe abschliessen">
  <ActorChoice name = "Weitere Bücher ausleihen">
    <Description>
      Aktor teilt dem System mit, dass er ein weiteres Buch ausleihen moechte
    </Description>
    <SystemStep>Leserkonto pruefen</SystemStep>
  </ActorChoice>
  <ActorChoice name = "Buchausleihe beenden">
    <Description>
      Aktor teilt dem System mit, dass er keine weiteren Buecher ausleihen moechte
    </Description>
    <Postcondition></Postcondition>
  </ActorChoice>
</ActorStep>

<Postcondition>
  <Condition>
    Mindestens ein Buch ist erfolgreich ausgeliehen worden.
  </Condition>
```

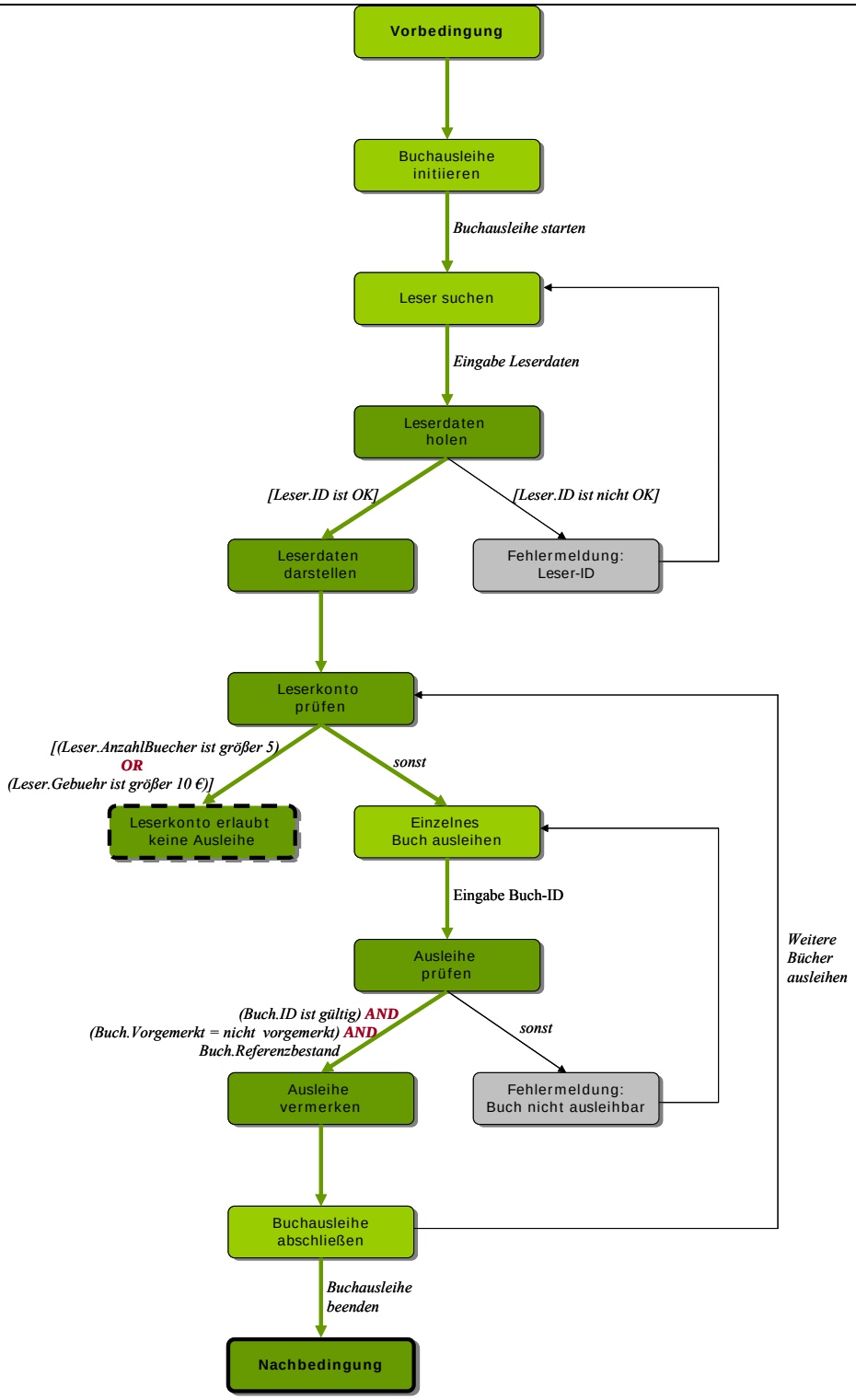
```
 </Postcondition>

 <Data name = "Leser">
  <Description>
   Ein Leser ist eine Person, die innerhalb der Biblbiothek Buecher ausleihen kann
  </Description>
  <Attribute name = "ID">
   <Description>
    Die ID eines Leser ist eine eindeutige Kennung des Lesers innerhalb der Bibliothek
   </Description>
  </Attribute>
  <Attribute name = "AnzahlBuecher">
   <Description>
    Beschreibt die aktuelle Anzahl ausgeliehener Buecher des Lesers
   </Description>
  </Attribute>
  <Attribute name = "Gebuehr">
   <Description>
    Beschreibt die aktuell vom Leser zu zahlende Gebühr (z.B. wegen nicht fristgemäßer
    Abgabe seine Buecher
   </Description>
  </Attribute>
 </Data>

 <Data name="Buch">
  <Description>
    Beschreibt eine ausleihbare Entität in der Bibliothek
  </Description>
  <Attribute name = "ID">
   <Description>
    Die ID identifiziert ein Buchexemplar eindeutigt innerhalb der Bibliothek
   </Description>
  </Attribute>
  <Attribute name = "Status">
   <Description>
    Beschreibt den aktuellen Status des Buches (z.B. ausgeliehen, Referenzbestand,
    vorgemerkt)
   </Description>
  </Attribute>
 </Data>

 <Exception name = "Leserkonto erlaubt keine Ausleihe">
  <Description>
   Der aktuelle Status des Leserkontos erlaubt keine Ausleihe. System gibt die Hinweise
   'Leserkonto erlaubt keine Ausleihe'.
  </Description>
 </Exception>
</UseCase>
```



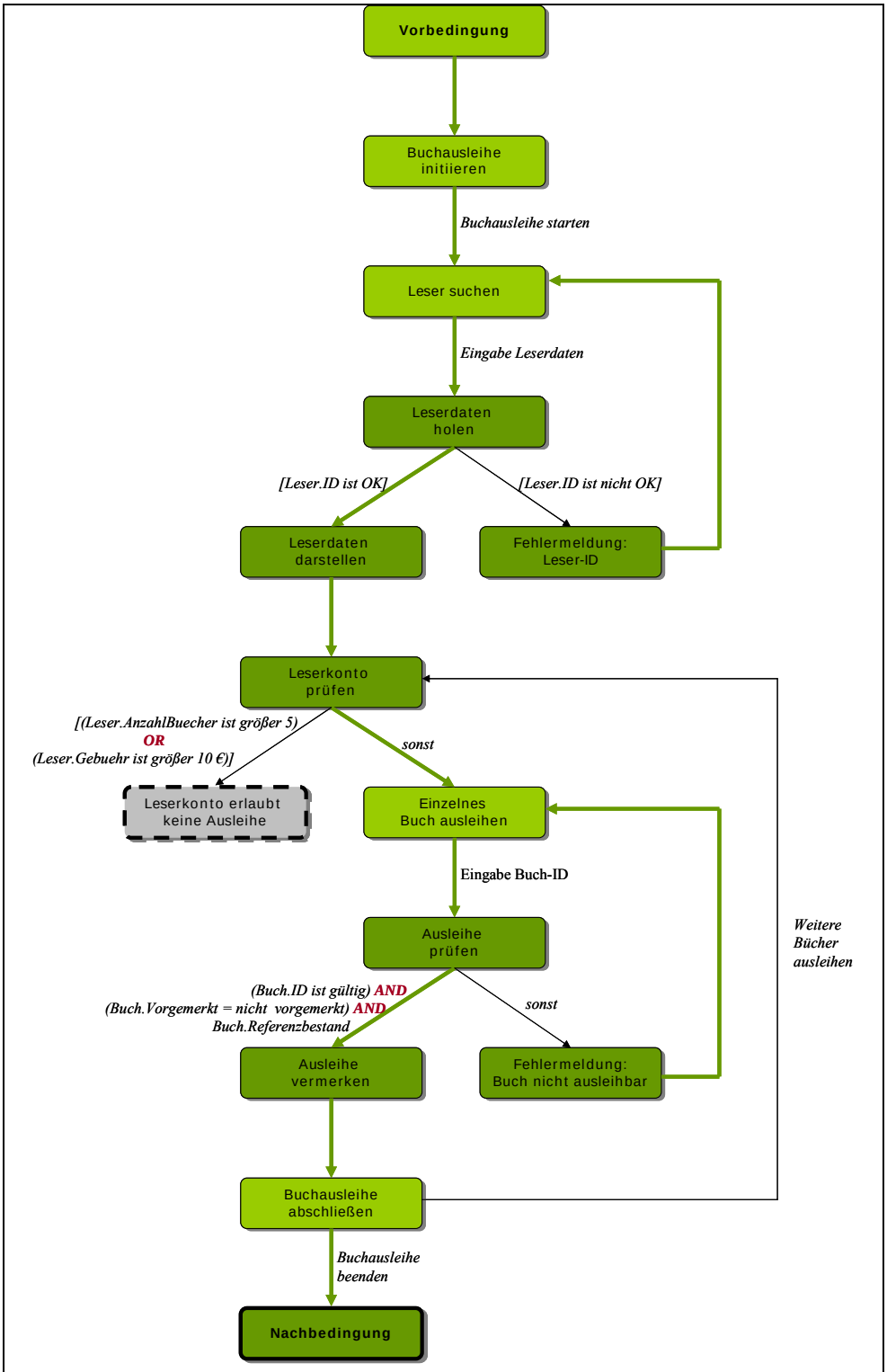
Anhang 2 – Testfälle des Kontrollflussgraphen für das Beispiels Überdeckungskriterium „Aktorschrittüberdeckung“ (Anzahl Testfälle: 1)



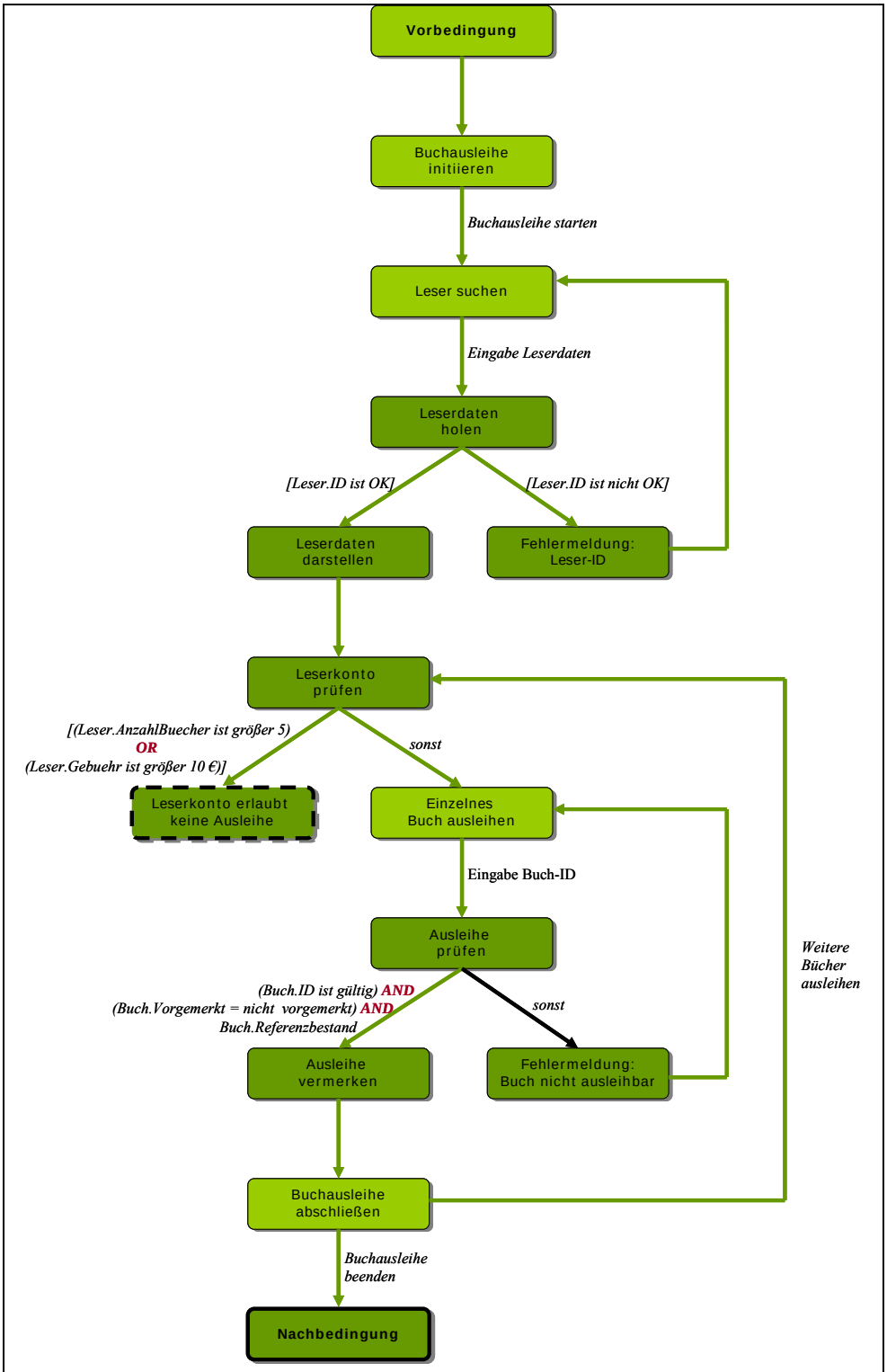
Anhang 3 – Testfälle des Kontrollflussgraphen für das Beispiels Überdeckungskriterium „Ausnahmeüberdeckung“ (Anzahl Testfälle: 1)



Anhang 4 – Testfälle des Kontrollflussgraphen für das Beispiels Überdeckungskriterium „Endknotenüberdeckung“ (Anzahl Testfälle: 2)



Anhang 5 – Testfälle des Kontrollflussgraphen für das Beispiels Überdeckungskriterium „Schrittüberdeckung“ (Anzahl Testfälle: 1)



Anhang 6 – Testfälle des Kontrollflussgraphen für das Beispiels Überdeckungskriterium „Kantenüberdeckung“ (Anzahl Testfälle: 2)

Leser						Buch					
Leser.ID		**Leser.AnzahlBuecher**		**Leser.Gebühr**		Buch.ID		Buch.Vorgemerkt		Buch.Referenzbestand	
ist OK	NICHT (ist OK)	größer 5	NICHT (größer 5)	größer 10 €	NICHT (größer 10 €)	ist gültig	NICHT (ist gültig)	true	false	true	false
X		X		X		X		X		X	
	X		X	X	X		X		X		X

Anhang 7 - Datenkombination des Beispiel-UCs „Bücherausleihe“ für die „each-used“ Überdeckung

Leser						Buch					
Leser.ID		**Leser.AnzahlBuecher**		**Leser.Gebühr**		Buch.ID		Buch.Vorgemerkt		Buch.Referenzbestand	
ist OK	NICHT (ist OK)	größer 5	NICHT (größer 5)	größer 10 €	NICHT (größer 10 €)	ist gültig	NICHT (ist gültig)	true	false	true	false
X		X		X		X		X		X	
	X		X	X			X		X	X	
X			X		X	X			X	X	
	X	X			X		X	X		*	*
X		X			X		X		X		X
	X		X	X		X		X			X

Anhang 8 - Datenkombination des Beispiel-UCs „Bücherausleihe“ für die paarweise Überdeckung