Reduktion von Testsuiten für Software-Produktlinien

Harald Cichos[1], Malte Lochau[2], Sebastian Oster[1], Andy Schürr[1]

[1]Technische Universität Darmstadt
Institut für Datentechnik
{cichos, oster, schuerr}@es.tu-darmstadt.de
[2]Technische Universität Braunschweig
Institut für Programmierung und Reaktive Systeme
lochau@ips.cs.tu-bs.de

Abstract: Eine Software-Produktlinie (SPL) bezeichnet eine Menge ähnlicher Produktvarianten, die bei entsprechend großer Anzahl einen erheblichen Testaufwand verursachen können. Viele modellbasierte SPL-Testansätze versuchen diesen Testaufwand zu verringern, indem Testfälle und Testmodelle aus vorangegangenen Testprozessen ähnlicher Produkte, wenn möglich, wiederverwendet werden. Eine weitere Möglichkeit den Testaufwand zu senken besteht darin, die Anzahl der auf den einzelnen Produkten auszuführenden Testfälle (Testsuite) mittels Testsuite-Reduktionstechniken zu reduzieren. Bisher existierende Verfahren wurden jedoch nicht für den Einsatz im SPL-Kontext entworfen und können daher nicht für jedes Produkt den Erhalt der erreichten Testabdeckung bzgl. eines Überdeckungskriteriums garantieren, wenn Testfälle produktübergreifend wiederverwendet werden. In dieser Arbeit wird diese aus der allgemeinen Testsuite-Reduktion bekannte Anforderung erstmals in erweiterter Form auf den SPL-Kontext übertragen. Darauf aufbauend werden zwei für SPLs ausgelegte Testsuite-Reduktionsansätze vorgestellt, die trotz Reduktion die erreichte Testabdeckung auf jedem Produkt beibehalten. Die Implementierung dieser Ansätze wird auf ein Anwendungsbeispiel angewendet und die Ergebnisse diskutiert.

1 Einleitung

Eine Software-Produktlinie (SPL) bezeichnet eine Menge sich in ihrer Funktionsweise ähnelnder Softwareprodukte [PBvdL05]. Die Ähnlichkeit der Softwareprodukte in dieser Menge resultiert aus der Verwendung einer gemeinsamen Programmplattform, welche mit zusätzlichen Programmteilen variabel erweitert wird. Diese variablen Programmteile realisieren funktionale Eigenschaften, sogenannte Features. Dabei führt eine hohe Anzahl an Features in der Regel zu einer ebenfalls hohen Anzahl an möglichen Produktvarianten. Wenn jede Produktvariante einer SPL ausführlich getestet werden soll, kann das bei großen SPLs, die viele Produkte umfassen, zu einem enormen Aufwand im Testprozess führen [McG01]. Diesen Aufwand versuchen viele modellbasierte SPL-Testansätze zu verringern, indem Testfälle und Testmodelle aus vorangegangenen Testprozessen ähnlicher Produktvarianten, wenn möglich, wiederverwendet werden. Eine weitere Möglichkeit den Aufwand zu verringern besteht darin, die auf den einzelnen Produkten auszuführenden

Testfälle (Testsuite) in ihrer Anzahl zu reduzieren. Existierende Testsuite-Reduktionstechniken sind jedoch nicht für den Einsatz im SPL-Kontext geeignet, da diese nur Testsuiten betrachten, deren Testfälle auf *einem* Produkt ausgeführt werden. In Testsuiten einer SPL können aber Testfälle existieren, die zur Ausführung auf *mehreren* Produkten vorgesehen sind [COLS11], um so die Kosten für Erstellung, Ausführung (z.B. Einsatz als Testorakel) oder Wartung (z.B. Testfallbeschreibung) zu senken. Aufgrund dieser produktübergreifenden Wiederverwendung von Testfällen können bisherige Reduktionstechniken den Erhalt der erreichten Testabdeckung bzgl. eines Überdeckungskriteriums nicht für jede Produktvariante garantieren [YH08].

In dieser Arbeit werden zwei für SPLs ausgelegte Testsuite-Reduktionsansätze vorgestellt, die trotz Reduktion die erreichte Testabdeckung auf jedem Produkt erhalten. Der erste Reduktionsansatz *entfernt* redundante Testfälle aus einer Testsuite für SPLs, wohingegen der zweite Ansatz eine Reduktion erreicht, indem dieser Paare von existierenden Testfällen durch jeweils einen einzelnen, neu erstellten Testfall *ersetzt*. Die Implementierung dieser Ansätze wird auf ein Anwendungsbeispiel angewendet und die Ergebnisse diskutiert. Beide Ansätze berücksichtigen die produktübergreifende Wiederverwendung von Testfällen und können daher selbst bei einer geringen Reduktion in der Testfallanzahl eine starke Reduktion in der Anzahl der Testfallausführungen erreichen. Beide Ansätze tragen dadurch zur Kostensenkung im Testprozess einer SPL bei.

2 Themenverwandte Arbeiten

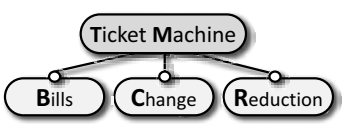
In der Regel nutzen SPL-Testansätze eine der folgenden zwei Strategien oder eine Kombination dieser [ER11]. Die erste Strategie verfolgt das Ziel die Anzahl der zu testenden Produkte zu reduzieren. Dafür wird aus den Produkten einer SPL eine Teilmenge identifiziert, die in Hinblick auf ein bestimmtes Kriterium repräsentativ für alle anderen Produkte ist. Anschließend wird die repräsentative Teilmenge stellvertretend für alle anderen Produkte getestet. Das von der repräsentativen Menge zu erfüllende Kriterium erfordert zumeist eine Überdeckung von Anforderungen [Sch07] oder basiert auf der Auswahl unterschiedlicher Kombinationen von Konfigurationsparametern (Features) der Produkte einer SPL, ähnlich wie beim kombinatorischen Test [OMR10]. Wie groß solch eine repräsentative Teilmenge ist, bzw. ob diese überhaupt in der SPL existiert, hängt maßgeblich von der Implementierung der zu testenden Produkte ab. Lässt sich keine repräsentative Teilmenge ausmachen, müssen im Rahmen der verfügbaren Ressourcen alle Produkte einer SPL einzeln getestet werden. Damit die zur Verfügung stehenden Ressourcen zum Testen einer großen Produktmenge ausreichen, verfolgt die zweite Strategie das Ziel, Testartefakte wie z.B. Testfälle oder Testmodelle für ähnliche Produkte wiederzuverwenden. Diese Ansätze nutzen teilweise Techniken des Regressionstests [ERS10], nur dass diese nicht auf die fortschreitenden Versionen eines einzelnen Produkts angewendet werden, sondern auf unterschiedliche Varianten eines Produkts. In [KBK11, COLS11] werden Ansätze vorgestellt, die für einen Testfall die Produkte ermitteln, auf denen dieser wiederverwendbar ist. In modellbasierten Testansätzen, wie z.B. SCenTED [RKPR05] und CADeT [Oli08] werden nicht die Testfälle, sondern die zur Herleitung von Testfällen genutzten Testmodel-

le wiederverwendet und für jedes Produkt angepasst. Eine ausführliche Zusammenfassung über modellbasierte Testansätze bietet [OWES11]. Von allen SPL-Testansätzen bisher völlig außer Acht gelassen wurde die Möglichkeit, die meist große Anzahl an auszuführenden Testfällen mittels Testsuite-Reduktionstechniken zu senken. Die vorliegende Arbeit nimmt sich als erste dieses Themas an und stellt zwei Ansätze vor, welche die in den Testfällen enthaltenen Informationen mit in den Reduktionsprozess einbeziehen, ähnlich wie in [CH11], um die erreicht Testabdeckung auf jedem Produkt zu erhalten. Eine ausführliche Übersicht über herkömmliche, nicht auf den SPL-Kontext zugeschnittene Testsuite-Reduktionstechniken bietet [YH08].

3 Grundlagen

In diesem Kapitel werden die grundlegenden Begriffe aus den Bereichen „Modellbasiertes Testen“ und „Software-Produktlinien“ anhand eines Anwendungsbeispiels, einer Fahrkartenautomaten-SPL (FA-SPL), eingeführt. Eine Software-Produktlinie (SPL) wird durch eine Menge von Softwareprodukten $PC = \{pc_1, pc_2, \ldots, pc_k\}$ beschrieben, die sich in ihrer Funktionsweise ähnlich sind. Alle Produkte einer SPL besitzen dieselbe Basisimplementierung und damit auch dieselbe Kernfunktionalität. In jeder dieser Produktvarianten wird diese Kernfunktionalität durch *Features* $F = \{f_1, f_2, \ldots, f_n\}$ individuell erweitert bzw. angepasst. Dabei gibt die jeweilige Produktkonfiguration vor, welche Features in einer Produktvariante verwendet werden. Es gilt $PC \subseteq \mathcal{P}(F)$. Beispielsweise bietet die FA-SPL insgesamt 4 Features $F_{FA} = \{TM, B, C, R\}$. Die zulässigen Feature-Kombinationen sind im Feature-Modell nach FODA [KCH$^+$90] in Abbildung 1(a) dargestellt. In Abbildung 1(b) sind die 8 aus dem Feature-Modell ableitbaren, zulässigen Produktkonfigurationen mit den genutzten Features aufgelistet. Es lässt sich erkennen, dass alle Produktkonfigurationen das Feature TM enthalten. Das Feature TM ist für die gemeinsame Kernfunktionalität verantwortlich, die sich wie folgt beschreiben lässt:

Zuerst wird die benötigte Anzahl an Fahrkarten ausgewählt. Im Gegensatz zur Tageskarte und Kurzstrecke lässt sich eine ermäßigte Tageskarte nur auswählen, wenn Feature R in der Produktkonfiguration enthalten ist. Anschließend startet der Bezahlvorgang, bei dem Münzen, und wenn Feature B enthalten ist, auch Scheine als Zahlungsmittel akzeptiert werden. Daraufhin erfolgt die Ausgabe der Fahrkarten. Abhängig von der Anwesenheit des Features C wird im nachfolgenden Schritt das Wechselgeld ausgegeben oder nicht.



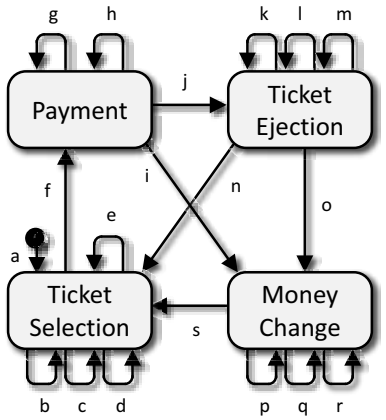
(a) Feature-Modell

	PC	1	2	3	4	5	6	7	8
Features*	TM	1	1	1	1	1	1	1	1
	B	0	0	0	0	1	1	1	1
	C	0	0	1	1	0	0	1	1
	R	0	1	0	1	0	1	0	1

* 1 = enthalten 0 = nicht enthalten

(b) Produktkonfigurationen

Abbildung 1: Zulässige Produktkonfigurationen der FA-SPL



(a) Zustandsautomat

#	F	Signal [Bedingung] / Aktion
a		/ costs=0;paid=0;tShort=0;tDay=0;tRed=0;
b		shortT / tShort++;costs+=2;
c		dayT / tDay++;costs+=4;
d	R	redT / tRed++;costs+=3;
e		cancel / tShort=0;tDay=0;tRed=0;
f		next [costs>0]
g		coin [paid<costs] / paid++;
h	B	bill [paid<costs] / paid+=5;
i		cancel [paid<costs] / tShort=0;tDay=0;tRed=0;
j		[paid>=costs] / paid-=costs;
k		[tShort>0 && tRed==0] / tShort--
l		[tDay>0 && tShort==0 && tRed==0] / tDay--
m	R	[tRed>0] / tRed--
n	¬C	[tDay==0 && tShort==0 && tRed==0] / paid=0;costs=0;
o	C	[tDay==0 && tShort==0 && tRed==0]
p	¬B	[paid>0] / paid--;
q	B	[paid>=5] / paid-=5;
r	B	[paid>0 && paid<5] / paid--;
s		[paid==0] / costs=0;

(b) Beschriftung der Transitionen mit benötigten Features

Abbildung 2: 150%-Testmodell der FA-SPL

Im modellbasierten Test werden die Testfälle, die auf das zu testende Softwareprodukt angewendet werden, aus einem *Testmodell* hergeleitet [UL06]. Das Testmodell beschreibt abstrakt das Verhalten des zu testenden Produkts und entspricht in dieser Arbeit einem Zustandsautomat, der aus Zuständen und Transitionen besteht. Im modellbasierten SPL-Test besitzt jedes zu testende Produkt pc ein zugehöriges Testmodell tm_{pc}. Alle produktspezifischen Testmodelle einer SPL lassen sich zu einer Menge von Testmodellen $TM = \{tm_1, \ldots, tm_k\}$ zusammenfassen. Zwecks einer kompakteren und übersichtlicheren Darstellung bietet es sich an, all diese produktspezifischen Testmodelle in einem sogenannten 150%-Testmodell [GKPR08] zusammenzufassen. Beispielsweise lassen sich alle 8 produktspezifischen Testmodelle der FA-SPL zu dem in Abbildung 2 dargestellten 150%-Testmodell vereinen. In Abbildung 2(a) ist die Struktur des 150%-Testmodells mit allen möglichen Zuständen und Transitionen dargestellt. Jede Transition ist mit einem Kleinbuchstaben gekennzeichnet, der auf die passende Transitionsbeschriftung in Abbildung 2(b) verweist. Um eines der 8 produktspezifischen Testmodelle tm_{pc} aus dem 150%-Testmodell herzuleiten, dürfen nur die Transitionen und Zustände aus dem 150%-Testmodell übernommen werden, die gemäß Produktkonfiguration pc zulässig sind. So darf eine Transition nur dann in ein Testmodell tm_{pc} übernommen werden, wenn die für die Transition benötigten Features (siehe Spalte F in Abbildung 2(b)) in dessen Produktkonfiguration pc enthalten sind. Beispielsweise werden für das Testmodell von Produkt 7 mit der Produktkonfiguration (TM,B,C) alle Transitionen bis auf d, m, n und p aus dem 150%-Testmodell übernommen (siehe Abbildung 3). Somit besteht jedes produktspezifische Testmodell aus einer Teilmenge von Zuständen und Transitionen des 150%-Testmodells.
Als Testendekriterium wird im modellbasierten Test oft ein Überdeckungskriterium (z.B. all-transitions oder all-states) auf das Testmodell angewendet. Dadurch werden Testziele $G = \{g_1, \ldots, g_l\}$ auf dem Testmodell definiert, die von den Testfällen erfüllt werden

PC	Exisitierende Testziele im jeweiligen produktspezifischen Testmodell																		
	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s
1	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓		✓		✓			✓
2	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓	✓		✓			✓
3	✓	✓	✓		✓	✓	✓		✓	✓	✓	✓			✓	✓			✓
4	✓	✓	✓	✓	✓	✓	✓		✓	✓	✓	✓	✓		✓	✓			✓
5	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓		✓			✓	✓	✓
6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			✓	✓	✓
7	✓	✓	✓		✓	✓	✓	✓	✓	✓	✓	✓			✓		✓	✓	✓
8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓		✓	✓	✓

Abbildung 3: Alle Produktkonfigurationen der FA-SPL und deren enthaltene Testziele (Transitionen)

müssen. In dieser Arbeit ergibt sich die Menge aller Testziele G aus dem 150%-Testmodell und somit implizit aus allen produktspezifischen Testmodellen. Wird beispielsweise das Überdeckungskriterium *all-transitions* auf das 150%-Testmodell der FA-SPL angewendet, stellt jede Transition eines jeden Produkts ein Testziel dar, das durch mindestens einen Testfall abgedeckt werden muss. Welche Testziele (Transitionen) in welchem produktspezifischen Testmodell enthalten sind, ist in Abbildung 3 dargestellt.
Im modellbasierten Test wird eine Menge von Testfällen $T = \{t_1, t_2, \ldots, t_m\}$ (*Testsuite*) für ein Produkt pc aus dem zum Produkt gehörenden Testmodell tm_{pc} hergeleitet. Jeder Testfall besteht dabei aus einer Sequenz von Eingaben und erwarteten Ausgaben. Werden die Eingaben auf das Testmodell tm_{pc} angewendet, lässt sich ein Testfall in einen Transitionspfad überführen, der aufzeigt, welche strukturellen Testziele vom Testfall in tm_{pc} abgedeckt werden. Nun existieren im modellbasierten SPL-Test mehrere produktspezifische Testmodelle. Um nicht für jede Produktvariante nahezu dieselben Testfälle wie für die Produktvarianten zuvor zu erstellen, wird überprüft, welche der bereits existierenden Testfälle wiederverwendet werden können. Ein Testfall t, hergeleitet aus Testmodell tm_{pc_i} für Produkt pc_i, lässt sich auch für Produkt pc_j wiederverwenden, g.d.w. die Eingabesequenz von t auf Testmodell tm_{pc_j} einen Transitionspfad erzeugt der genau dieselben Testziele abdeckt wie in tm_{pc_i}. In diesem Fall ist t sowohl auf Produkt pc_i, als auch auf pc_j *valide ausführbar*. Eine entsprechende Relation *valid* $\subseteq T \times G \times PC$ ist wie folgt definiert:

valid$(t, g, pc) :\Leftrightarrow$ Testfall t ist *valide ausführbar* auf Produktkonfiguration pc, wenn der Transitionspfad von t auf tm_{pc} das Testziel g erreicht.

Auf welchen Produktkonfigurationen ein Testfall valide ausführbar ist, lässt sich durch den in [COLS11] vorgestellten Ansatz feststellen. Mit diesem Ansatz ist es auch möglich, eine *vollständige SPL-Testsuite* auf effiziente Weise herzuleiten. Eine vollständige SPL-Testsuite besteht aus Testfällen, die auf jedem produktspezifischen Testmodell einer SPL eine vollständige Überdeckung für ein gewähltes Überdeckungskriterium C erreichen. Eine vollständige SPL-Testsuite ist in [COLS11] wie folgt definiert:

Definition 1 *(Vollständige SPL-Testsuite)*
Eine SPL-Testsuite $TS_C = (T', PC')$ mit einer Menge von Testfällen $T' \subseteq T$ und zulässigen Produktkonfigurationen $PC' \subseteq PC$ ist vollständig *bzgl. des Überdeckungskriteriums C, das die Testziele G definiert, gdw.:*

$$\forall g \in G, pc \in PC: \quad (\exists t \in T : \textit{valid}(t, g, pc)) \Rightarrow (\exists t' \in T' : \textit{valid}(t', g, pc))$$

Testfall	Features*				Transitionspfad
	TM	B	C	R	
t1	1	-	-	-	a b f
t2	1	1	0	1	a e d f h j m n
t3	1	-	-	-	a e
t4	1	-	-	-	a c f g g g g j l
t5	1	-	-	1	a d f g g g j m
t6	1	-	0	-	a b f g g j k n
t7	1	-	1	-	a b f g g j k o
t8	1	0	-	-	a b f g i p
t9	1	1	-	-	a b c f h i q
t10	1	1	-	-	a b f g i r
t11	1	-	-	-	a b f i s

* 1 = enthalten 0 = nicht enthalten - = egal

(a) SPL-Testsuite

PC	Testsuite	Anzahl
1	t3, t4, t6, t8, t11	5
2	t3, t4, t5, t6, t8, t11	6
3	t3, t4, t7, t8, t11	5
4	t3, t4, t5, t7, t8, t11	6
5	t3, t4, t6, t9, t10, t11	6
6	t3, t4, t5, t6, t9, t10, t11	7
7	t3, t4, t7, t9, t10, t11	6
8	t3, t4, t5, t7, t9, t10, t11	7
		= 48

(b) Produktspez. Testsuites (minimal)

Abbildung 4: FA-SPL-Testsuite ohne redundante Testfälle

Dabei bezeichnet $PC' \subseteq PC$ die Teilmenge der Produktmenge der SPL, die zum Testen ausgewählt wurde. In dieser Arbeit wird von $PC' = PC$ ausgegangen. Die SPL-Testsuite der FA-SPL ist in Abbildung 4(a) dargestellt. Sie enthält 11 wiederverwendbare Testfälle. Für jeden Testfall sind die Features, die dieser zur validen Ausführung auf einem Produkt benötigt, angegeben. Die von einem Testfall abgedeckten Testziele (Transitionen) sind aus dessen Transitionspfad ermittelbar. Aus dieser SPL-Testsuite lässt sich für jedes Produkt der FA-SPL eine produktspezifische Testsuite bilden, die auf dem dazu passenden Testmodell eine vollständige Überdeckung erreicht (siehe Abbildung 4(b)). Es ist zu beachten, dass ein auf einem Produkt valide ausführbarer Testfall nicht zwangsläufig auch auf diesem ausgeführt werden muss. Beispielsweise ist Testfall $t1$, obwohl dieser auf allen 8 Produkten valide ausführbar ist, auf keinem Produkt zur Ausführung eingeplant.

4 Reduktion von Testsuiten für Software-Produktlinien

Die Testsuite-Reduktion bezeichnet im Allgemeinen einen Vorgang, bei dem die Anzahl der in einer Testsuite enthaltenen Testfälle reduziert wird, um die Kosten bei der Testfallausführung zu senken. In einer SPL-Testsuite kann, aufgrund der möglichen Wiederverwendung von Testfällen, die Anzahl der enthaltenen Testfälle geringer ausfallen als die Anzahl der benötigten Testfallausführungen. Beispielsweise enthält die SPL-Testsuite der FA-SPL in Abbildung 4(a) nur 11 Testfälle, obwohl 48 Testfallausführungen benötigt werden, um auf allen 8 Produkten eine all-transitions-Überdeckung zu erreichen (siehe Abbildung 4(b)). Diese Entkopplung hat auch Auswirkungen auf die Reduktion. So kann auch eine nur geringfügig reduzierte SPL-Testsuite eine große Reduktion in der Anzahl der Testfallausführungen nach sich ziehen. Aus diesem Grund sollte bei der Testsuite-Reduktion im SPL-Kontext nicht die Anzahl der reduzierten Testfälle im Vordergrund stehen, sondern die Anzahl der reduzierten Testfallausführungen.
Existierende Testsuite-Reduktionstechniken gehen von der Annahme aus, dass die Testfälle einer Testsuite nur auf einem *einzigen* Produkt ausgeführt werden. Aus diesem Grund erfüllen diese Ansätze lediglich folgende Forderung:

Bisherige Forderung: *Bei der Reduktion einer Testsuite darf der durch diese Testsuite erzielte Grad der Abdeckung bezüglich eines geforderten Abdeckungskriteriums nicht verringert werden.*

Da im SPL-Kontext die Testfälle nicht nur durch die Menge abgedeckter Testziele charakterisiert sind, sondern auch durch die Menge von Produktvarianten, auf denen diese valide ausführbar sind, muss diese Forderung für SPLs wie folgt erweitert werden:

Neue Forderung: *Bei der Reduktion einer SPL-Testsuite darf der durch diese Testsuite erzielte Grad der Abdeckung bezüglich eines geforderten Abdeckungskriteriums auf den zu testenden Produkten der SPL nicht verringert werden.*

Wird nur der bisherigen Forderung bei der Reduktion einer SPL-Testsuite nachgekommen, kann der Erhalt der erreichten Testabdeckung nicht für jedes Produkt garantiert werden. Beispielweise könnten bisherige Ansätze fälschlicherweise den Testfall $t3$ der FA-SPL entfernen (siehe Abbildung 4), in der Annahme, dass dessen abgedeckten Testziele auch noch von Testfall $t2$ abgedeckt werden. Dass Testfall $t2$ aber nur auf Produkt 6 und nicht wie $t3$ auf allen 8 Produkten ausführbar ist, würde nicht bemerkt werden. Somit wäre das Testziel e auf 7 von 8 Produkten durch keinen Testfall mehr abgedeckt.

4.1 Entfernen redundanter Testfälle

Eine Möglichkeit, die Anzahl der Testfälle zu reduzieren und zugleich der neuen Forderung nachzukommen, besteht im gezielten Entfernen redundanter Testfälle aus einer SPL-Testsuite T. Ein Testfall t ist *redundant*, wenn es einen Testfall $t' \in T$, $t \neq t'$ gibt, der t *subsumiert*. Das heißt, t' deckt alle Testziele von t auf den gleichen Produkten wie t ab. Beispielsweise wird in Abbildung 4(a) der Testfall $t1$ von Testfall $t11$ subsumiert. Wir definieren hierfür eine Subsumptionsrelation $t \preceq t'$ auf Testfallpaaren $t, t' \in T$ eines 150%-Testmodells, einer Menge von Testzielen G und einer Menge von zu testenden Produktkonfigurationen PC wie folgt:

$$t \preceq t' :\Leftrightarrow \forall g \in G, pc \in PC : (\mathit{valid}(t, g, pc) \Rightarrow \mathit{valid}(t', g, pc))$$

In der Regel sind redundante Testfälle in einer Testsuite nicht vollständig auf Subsumption zwischen Paaren von Testfällen zurückführbar. Vielmehr können Subsumptionsbeziehungen auch zwischen *Mengen* von Testfällen der Testsuite bestehen. Beispielsweise wird in Abbildung 4(a) der Testfall $t2$ nicht von einem einzelnen Testfall, sondern von einer Testfallmenge, bestehend aus $t3$, $t5$, $t6$ und $t9$, subsumiert. Daher wird der Subsumptionsbegriff durch die folgende Definition weiter verallgemeinert.

Definition 2 *(SPL-Testsuite Subsumption)*
Eine Testsuite-Subsumptionsrelation $\preceq \subseteq \mathcal{P}(T) \times \mathcal{P}(T)$ *ist eine Ordnung auf Teilmengen von Testfällen einer Testsuite T, sodass:*

$$T' \preceq T'' :\Leftrightarrow \forall g \in G, pc \in PC : (\exists t' \in T' : \mathit{valid}(t', g, pc)) \Rightarrow (\exists t'' \in T'' : \mathit{valid}(t'', g, pc))$$

Durch den nun definierten Subsumptionsbegriff ergibt sich folgender Satz:

Satz 1 *(SPL-Testsuite-Reduktion)*
Wenn $TS' = (T', PC')$ eine vollständige SPL-Testsuite ist, dann ist $TS'' = (T'', PC')$ auch eine vollständige SPL-Testsuite, falls $T' \preceq T''$ gilt.

Beweis: Folgt direkt aus Definition 2.

Folglich kann die Anzahl der Testfälle einer SPL-Testsuite mit Hilfe des eingeführten Subsumptionsbegriffes um redundante Testfälle reduziert werden, ohne die SPL-Abdeckung zu verringern.

4.1.1 Algorithmus zum Entfernen redundanter Testfälle

Im Folgenden wird der in Abbildung 5 dargestellte Algorithmus zum Entfernen aller redundanten Testfälle aus einer SPL-Testsuite näher erläutert. Dieser Algorithmus überprüft jeden Testfall t in einer SPL-Testsuite $splTS$ daraufhin (1.), ob t ein Testziel (3.) auf einem Produkt, auf dem t ausführbar ist (4.), als einziger Testfall abdeckt (5.). Um effizient zu ermitteln, ob noch ein weiterer Testfall existiert der Testziel g auf Produkt pc abdeckt, wird in $splCoverage$ für jedes Testziel auf jedem Produkt protokolliert von wievielen Testfällen dieses abgedeckt wird (max. Speicherverbrauch: $|G| \cdot |PC|$). Testfall t ist nicht redundant (6.), wenn ein Testziel g auf einem Produkt pc nur von Testfall t abgedeckt wird. Sollten alle Testziele, die t abdeckt, auf allen Produkten, auf den t ausführbar ist, auch noch von anderen Testfällen abgedeckt werden, dann ist t redundant und kann aus der Testsuite entfernt werden (8.). Nach dem Entfernen des Testfalls muss $splCoverage$ für die nächste Iteration aktualisiert werden (9.). Wird dieser Algorithmus auf die FA-SPL angewendet (siehe Abbildung 4(a)), werden die Testfälle $t1$ und $t2$ entfernt. Der Algorithmus garantiert aber *nicht* die Herleitung einer minimalen SPL-Testsuite. Dafür muss die größtmögliche Anzahl an redundanten Testfällen aus einer SPL-Testsuite entfernt werden, was NP-schwer ist [HCL+03].

```
1.  for all t in splTS
2.      isRedundant = true;
3.      for all g in t.getSatisfiedGoals()
4.          for all pc in t.getPCs_t_isValidFor()
5.              if splCoverage.isOnlyOneWhichCovers(t,g,pc) then
6.                  isRedundant = false
7.      if isRedundant then
8.          splTS.remove(t)
9.          splCoverage.update()
```

Abbildung 5: Pseudocode zum Entfernen von redundanten Testfällen in einer SPL-Testsuite

4.2 Ersetzen von Testfällen

Enthält eine SPL-Testsuite keine redundanten Testfälle, lässt sich deren Testfallmenge T reduzieren, indem eine Menge von Testfällen $T' \subseteq T$ durch eine Menge neu erstellter Testfälle T'' ($T'' \cap T = \emptyset$) ersetzt wird, sodass gilt:

$$T' \preceq (T \setminus T') \cup T'' \text{ und somit auch } T \preceq (T \setminus T') \cup T''$$

Das heißt, wenn ein Testziel g eines Produkts pc *nur* von Testfällen aus T' abgedeckt wird, dann muss mindestens ein Testfall in T'' existieren, der g auf pc abdeckt. Da häufig noch Testfälle in $T \setminus T'$ existieren, die g in pc ebenfalls abdecken, sinkt die Anzahl der Testziele, die von den neuen Testfällen in T'' zwingend abgedeckt werden müssen. Damit durch das Ersetzen von Testfällen eine Reduktion erreicht wird, muss gelten:

$$\mid T' \mid > \mid T'' \mid \text{ und somit auch } \mid T \mid > \mid (T \setminus T') \cup T'' \mid$$

4.2.1 Algorithmus zum Ersetzen von Testfallpaaren

Im Folgenden wird der in Abbildung 6 dargestellte Algorithmus erläutert, der Testfallpaare in einer SPL-Testsuite durch jeweils einen neu erstellten Testfall ersetzt. Als Ausgangssituation wird angenommen, dass die zu reduzierende SPL-Testsuite keine redundanten Testfälle mehr enthält, da das Entfernen von redundanten Testfällen kostengünstiger ist als das Ersetzen, bei dem immer ein neuer Testfall erstellt wird.
Der Algorithmus versucht jeweils zwei Testfälle $t1$ und $t2$ (Testfallpaar) durch einen neu erstellten Testfall $tnew$ zu ersetzen. Dabei werden nur solche Paare betrachtet (1.), deren zwei Testfälle auf genau derselben Menge an Produkten valide ausführbar sind. Diese Forderung basiert auf der Erfahrung, dass ein neu erstellter Testfall häufig nur dann auf den Produkten des Testfallpaares valide ausführbar ist, wenn die beiden Testfälle des Testfallpaares bereits auf derselben Menge an Produkten valide ausführbar sind. Damit nach dem Ersetzungsschritt weiterhin jedes Testziel auf jedem Produkt abgedeckt wird, werden die Testziele bestimmt (2.), die nicht mehr abgedeckt werden würden, wenn das Testfallpaar ersatzlos aus der Testsuite entfernt wird. Für diese ausgewählten Testziele wird anschließend ein neuer Testfall $tnew$ für die Menge an Produkten gesucht, auf denen das

```
1.  for all pair(t1,t2) in splTS with pair.hasSameValidPCs(t1,t2)
2.      necessaryGoals = splCoverage.findUncoveredGoalsIfRemoved(t1,t2)
3.      tnew = createTestcase(necessaryGoals, getPCsPairIsValidFor(t1,t2))
4.      if tnew exists then
5.          pcs = analyzer.findPCsTestcaseIsValidFor(tnew)
6.          if getPCsPairIsValidFor(t1,t2) is subset of pcs then
7.              splTS.remove(t1)
8.              splTS.remove(t2)
9.              splTS.add(tnew)
10.             splCoverage.update()
```

Abbildung 6: Pseudocode zum Ersetzen von Testfallpaaren in einer SPL-Testsuite

	Testfall	Features*				Transitionspfad
		TM	B	C	R	
ersetzt durch t12	~~t3~~	1	-	-	-	a e
	~~t4~~	1	-	-	-	a c f g g g g j l
	t5	1	-	-	1	a d f g g g j m
	t6	1	-	0	-	a b f g g j k n
	t7	1	-	1	-	a b f g g j k o
ersetzt durch t13	t8	1	0	-	-	a b f g i p
	~~t9~~	1	1	-	-	a b c f h i q
	~~t10~~	1	1	-	-	a b f g i r
	t11	1	-	-	-	a b f i s
	t12	1	-	-	-	a e c f g g g g j l
	t13	1	1	-	-	a c c f g h i q r

* 1 = enthalten 0 = nicht enthalten - = egal

(a) SPL-Testsuite

PC	Testsuite	Anzahl
1	t6, t8, t11, t12	4
2	t5, t6, t8, t11, t12	5
3	t7, t8, t11, t12	4
4	t5, t7, t8, t11, t12	5
5	t6, t11, t12, t13	4
6	t5, t6, t11, t12, t13	5
7	t7, t11, t12, t13	4
8	t5, t7, t11, t12, t13	5
		= 36

(b) Produktspez. Testsuites (minimal)

Abbildung 7: Reduzierte FA-SPL-Testsuite aufgrund ersetzter Testfallpaare

Testfallpaar valide ausführbar ist (3.). Lässt sich ein solcher Testfall $tnew$ erstellen (4.), muss anschließend die Menge an Produkten ermittelt werden (5.), auf denen $tnew$ valide ausführbar ist. Für die Herleitung (3.) und Auswertung (5.) des Testfalls $tnew$ kann beispielsweise der Ansatz aus [COLS11] benutzt werden. Wenn der neue Testfall $tnew$ mindestens auf denselben Produkten valide ausführbar ist wie das Testfallpaar (6.), kann Testfall $tnew$ das Testfallpaar ersetzen ohne die SPL-Abdeckung zu senken. Anschließend kann das Testfallpaar entfernt (7.-8.) und der Testfall $tnew$ in die SPL-Testsuite aufgenommen werden (9.). Nachdem die SPL-Testsuite verändert wurde, muss die SPL-Abdeckung $splCoverage$ für die nächste Iteration aktualisiert werden. Durch jeden Ersetzungsschritt reduziert sich die Anzahl der in der SPL-Testsuite $splTS$ enthaltenen Testfälle um 1. Wird dieser Algorithmus auf die FA-SPL angewendet (siehe Abbildung 7), wird das Testfallpaar $t3$ und $t4$ durch $t12$ und das Testfallpaar $t9$ und $t10$ durch $t13$ ersetzt. Dadurch ergibt sich, ausgehend von einer redundanzfreien FA-SPL-Testuite, eine Reduktion in der Anzahl der Testfälle von 9 auf 7 und eine Reduktion in der Anzahl der Testfallausführungen von 48 auf 36 (vergleiche Abbildung 4(b) mit 7(b)). Die stark reduzierte Anzahl der Testfallausführungen um 25% lässt sich darauf zurückführen, dass beide Testfallpaare auf vielen Produkten der FA-SPL zur Ausführung vorgesehen waren.

4.3 Diskussion

Im Rahmen dieser Arbeit wurden die beiden vorgestellten Ansätze in Azmun [Has], ein Framework für den modellbasierten Test, realisiert und auf weitere, aus 150%-Testmodellen hergeleitete Testsuiten angewendet. Die Auswertung der untersuchten SPL-Testsuiten ergab, dass die Anzahl an Testfallausführungen durch Ersetzen von Testfällen umso stärker reduziert werden kann, je *größer* die gemeinsame Menge von Produkten ist, auf der die zu ersetzenden Testfälle zur Ausführung eingeplant sind. Folglich lässt sich der in Abschnitt 4.2.1 vorgestellte Algorithmus in seiner Effizienz steigern, wenn die Paare zuvor noch nach der Anzahl an Produkten, auf denen diese beide zur Ausführung vorgesehen sind, sortiert werden.

Bei der Reduktion durch Ersetzen ist zu beachten, dass die Produktmenge, auf der sich ein neu erstellter Testfall ausführen lässt, häufig (abhängig vom 150%-Testmodell) umso kleiner ausfällt, je mehr Testziele dieser zwingend abdecken muss. Sollte aber die Produktmenge eines solchen Testfalls nicht eine Obermenge der Produktmenge sein, auf welcher die zu ersetzenden Testfälle valide ausführbar sind, ist eine Subsumption nicht mehr möglich. Aus diesem Grund sollte bei der Erstellung eines Testfalls darauf geachtet werden, dass von diesem nur die Abdeckung der zwingend notwendigen Testziele gefordert wird. Eine Methode zur Bestimmung dieser Testziele wurde in Abschnitt 4.2 vorgestellt.

Beide vorgestellten Ansätze reduzieren eine SPL-Testsuite produktübergreifend. Jedoch ist es auch möglich, jede produktspezifische Testsuite einer SPL einzeln zu reduzieren, solange die in Abschnitt 4.1 neu eingeführte Forderung bzgl. der SPL-Abdeckung erfüllt wird. Dieses Vorgehen wirkt sich allerdings negativ auf die Wiederverwendbarkeit der Testfälle aus, sowohl beim Entfernen als auch beim Ersetzen. So sinkt beim Entfernen von Testfällen zwar die Anzahl der Ausführungen, aber nicht zwangsweise die Anzahl der in einer SPL-Testsuite enthaltenen Testfälle. Beispielsweise könnte in jeder produktspezifischen Testsuite ein anderer redundanter Testfall entfernt werden. Beim Ersetzen kommt hinzu, dass die neuen Testfälle für jede produktspezifische Testsuite erneut erstellt werden, was einerseits zu hohen Kosten in der Testfallgenerierung führt und andererseits zu evtl. nicht wiederverwendbaren Testfällen. Durch Letzteres steigt die Anzahl der in der SPL-Testsuite enthaltenen Testfälle, was wiederum zu höheren Wartungskosten bei den Testfällen führt (z.B. Aktualisierung der Testfallbeschreibung). Testfälle sollten auch nur dann zwecks Kosteneinsparung ersetzt werden, wenn Grund zur Annahme besteht, dass durch das Erstellen *und* die Ausführung der neu erstellten Testfälle weniger Kosten als durch die Ausführung der zu ersetzenden Testfälle entstehen.

Beide vorgestellten Reduktionsansätze garantieren nur den Erhalt der erreichten Überdeckung einer SPL-Testsuite, aber nicht den Erhalt der erreichten Gesamtfehlersensitivität einer Testsuite. Aus diesem Grund sollten beide Ansätze nur dann angewendet werden, wenn eine SPL-Testsuite aufgrund von Kostengründen reduziert werden *muss* und dabei keine Rücksicht auf den Erhalt der Gesamtfehlersensitivität genommen werden kann.

5 Zusammenfassung

In dieser Arbeit wurden zwei neue Testsuite-Reduktionsansätze für SPLs vorgestellt. Im Gegensatz zu bisher existierenden Testsuite-Reduktionstechniken berücksichtigen beide Ansätze, dass im SPL-Kontext Testfälle existieren können, die zur Ausführung auf mehreren Produkten vorgesehen sind. Dadurch ist es mit beiden Ansätzen möglich, die Anzahl der in einer SPL-Testsuite enthaltenen Testfälle produktübergreifend zu reduzieren ohne dabei die zuvor erreichte Testabdeckung auf einem einzigen Produkt zu verringern. Aufgrund dieser produktübergreifenden Testsuite-Reduktion bleiben die verbleibenden Testfälle in einer SPL-Testsuite wiederverwendbar. Für zukünftige Arbeiten ist eine Evaluation beider Ansätze an industriellen Beispielen geplant, die zeigen soll, ob diese skalieren und in der Praxis anwendbar sind.

Literatur

[CH11] H. Cichos und T. Heinze. Efficient Test Suite Reduction by Merging Pairs of Suitable Test Cases. In J. Dingel und A. Solberg, Hrsg., *Models in Software Engineering - Workshops and Symposia at MODELS 2010*, Jgg. 6627 of *LNCS*, Seiten 244–258, 2011.

[COLS11] H. Cichos, S. Oster, M. Lochau und A. Schürr. Model-based Coverage-Driven Test Suite Generation for Software Product Lines. In *Proc. of MoDELS'2011*, Jgg. 6981 of *LNCS*, Seiten 425–439, 2011.

[ER11] E. Engström und P. Runeson. Software Product Line Testing - A Systematic Mapping Study. *Information and Software Technology*, 50:1098–1113, January 2011.

[ERS10] E. Engström, P. Runeson und M. Skoglund. A Systematic Review on Regression Test Selection Techniques. *Information & Software Technology*, 52(1):14–30, 2010.

[GKPR08] H. Grönniger, H. Krahn, C. Pinkernell und B. Rumpe. Modeling Variants of Automotive Systems using Views. In *Modellierung*, 2008.

[Has] S. Haschemi. Azmun - The Model-Based Testing Framework. Online: 2011-10-05.

[HCL+03] H. S. Hong, S. D. Cha, I. Lee, O. Sokolsky und H. Ural. Data Flow Testing as Model Checking. In *Proc. of ICSE'03*, Seiten 232–242. IEEE, 2003.

[KBK11] C.H.P. Kim, D.S. Batory und S. Khurshid. Reducing Combinatorics in Testing Product Lines. In *Proc. of AOSD'2011*, Seiten 57–68. ACM, 2011.

[KCH+90] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson. Feature-Oriented Domain Analysis (FODA). Bericht, Carnegie-Mellon University, 1990.

[McG01] J. D. McGregor. Testing a Software Product Line. Bericht CMU/SEI-2001-TR-022, Carnegie Mellon, Software Engineering Institute, 2001.

[Oli08] E. M. Olimpiew. *Model-Based Testing for Software Product Lines*. Dissertation, George Mason University, 2008.

[OMR10] S. Oster, F. Markert und P. Ritter. Automated Incremental Pairwise Testing of Software Product Lines. In *Proc. of SPLC'2010*, Seiten 196–210, 2010.

[OWES11] S. Oster, A. Wübbeke, G. Engels und A. Schürr. Model-Based Software Product Lines Testing Survey. In J. Zander, I. Schieferdecker und P. Mosterman, Hrsg., *Model-based Testing for Embedded Systems*. CRC Press/Taylor&Francis, 2011.

[PBvdL05] K. Pohl, G. Böckle und F. van der Linden. *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer, 2005.

[RKPR05] A. Reuys, E. Kamsties, K. Pohl und S. Reis. Model-based System Testing of Software Product Families. In *Proc. of CAiSE'2005*, Seiten 519–534, 2005.

[Sch07] K. Scheidemann. *Verifying Families of System Configurations*. Dissertation, TU Munich, 2007.

[UL06] M. Utting und B. Legeard. *Practical Model-Based Testing: A Tools Approach*. Morgan Kaufmann Publishers Inc., 2006.

[YH08] S. Yoo und M. Harman. Regression Testing Minimization, Selection and Prioritization: A Survey. *Software Testing, Verification and Reliability*, 2008.