Fraunhofer
IESE

D-5.1: Ansatz für konditionale Testfallreduktion basierend auf vorausgegangenen Testergebnissen

Autoren:
Thomas Bauer
Robert Eschbach
Zhensheng Guo
Johannes Kloos

Projekt ranTEST
Förderkennzeichen: 01 IS E09
Forschungsoffensive
„Software Engineering 2006“

IESE-Report Nr. 044.09/D
Version 1.0
Dezember 2008

Eine Publikation des Fraunhofer IESE

*Das ranTEST-Konsortium besteht aus Fraunhofer IESE, Universität Duisburg-Essen,
Siemens AG Transportation Systems und Market maker Software AG.

Das Fraunhofer IESE ist ein Institut der Fraunhofer-Gesellschaft.
Das Institut transferiert innovative Software-Entwicklungstechniken, -Methoden und -Werkzeuge in die industrielle Praxis. Es hilft Unternehmen, bedarfsgerechte Software-Kompetenzen aufzubauen und eine wettbewerbsfähige Marktposition zu erlangen.

Das Fraunhofer IESE steht unter der Leitung von
Prof. Dr. Dieter Rombach (geschäftsführend)
Prof. Dr. Peter Liggesmeyer
Fraunhofer-Platz 1
67663 Kaiserslautern

Abstract

Dieses Dokument ist ein Bericht zu AP-5.1. Es betrachtet Verfahren zur bedingten Durchführung von Testfällen unter Verwendung des Wissens, das aus der vorherigen Ausführung anderer Testfälle gewonnen wurde. Hierzu wird ein Verfahren entwickelt, das während der Ausführung der Testsuite solche Testfälle erkennt, die keine weitere Information liefern würden, und deren Ausführung unterbindet.

Schlagworte: Test suite reduction, regression testing, risk-based testing, quality metrics, ranTEST

Inhaltsverzeichnis

Abstractv

1	**Einführung**	**1**
2	**Ziele und Vorgehen**	**2**
2.1	Ziele der konditionalen Testfallreduktion	2
2.2	Vorgehen	2
3	**Stand der Technik**	**3**
3.1	Kosten-/Nutzen-Analyse von Techniken zur konditionalen Testausführung	3
3.2	Ansätze aus der Testsuite-Reduktion	3
3.3	Ansätze aus dem Gebiet des Regressionstests	4
4	**Abhängigkeitsanalyse**	**8**
4.1	Bewertungskriterien der Testergebnisse	8
4.2	Abhängigkeiten	8
5	**Algorithmen zur konditionalen Testfallreduktion**	**10**
5.1	Der Präfix-Baum	10
5.2	Algorithmen	12
5.2.1	Konstruktion des Präfix-Baums	12
5.2.2	Bedingte Testfallausführung: automatische Testfall-Auswahl ohne Priorisierung	13
5.2.3	Wiederholung der Testfallausführung nach Fehlerkorrektur	17
5.2.4	Mögliche Erweiterungen	18
6	**Bewertung**	**19**
7	**Zusammenfassung**	**21**

1 Einführung

Inhalt von D5.1 ist das Thema „konditionale Testfallreduktion basierend auf vorausgegangenen Testergebnissen“. Hierbei sollen die Ergebnisse der Durchführung von Testfällen an Kontrollpunkten bewertet werden, um die weitere Testplanung zu beeinflussen. Weiterhin werden Kriterien untersucht, die die Beendigung des Tests gestatten.

Zunächst werden in Kapitel 2 kurz die Ziele dieses Arbeitspakets dargestellt. Danach werden wir in Kapitel 3 den bisherigen Stand der Forschung untersuchen, wobei wir insbesondere auf Literatur aus der Testfall-Reduktion und aus dem Regressionstesten eingehen. Davon ausgehend leiten wir in Kapitel 4 ein Kriterium her, mit dessen Hilfe wir aus den Ergebnissen bereits ausgeführter Testfälle ermitteln, welche weiteren Testfälle ausgeführt werden müssen. Die effiziente Implementierung dieses Kriteriums ist Inhalt von Kapitel 5. In Kapitel 6 findet sich eine Bewertung des Verfahrens, und in Kapitel 7 sind unsere Ergebnisse zusammengefasst.

2 Ziele und Vorgehen

2.1 Ziele der konditionalen Testfallreduktion

Das Ziel ist, ein Verfahren zu erarbeiten, das an gewissen Kontrollpunkten während der Testdurchführung die bereits vorliegenden Testergebnisse auswertet, um die überflüssige Durchführung anderer Testfälle zu vermeiden. Hierzu soll erkannt werden, welche noch nicht durchgeführten Testfälle keine bzw. keine relevanten neuen Informationen liefern würden.

2.2 Vorgehen

Dieser Text gliedert sich in folgende Schritte:

- Sichtung des Stands der Technik.
- Erforschung der Abhängigkeiten zwischen Testergebnissen und weiteren Testfällen. Hier werden wir insbesondere auf Abhängigkeiten zwischen Testfällen aufgrund gemeinsamer Folgen von Testschritten eingehen.
- Vorschlag eines Verfahrens zur Erzeugung eines Abhängigkeitsmodells. Ausgehend von dem oben ermittelten Abhängigkeitskriterium werden wir einen Algorithmus angeben, der in effizienter Weise ermöglicht, ein Abhängigkeitsmodell zu erzeugen und davon ausgehend eine Menge von Tests aus einem Testvorgang herauszunehmen, wenn diese keine neuen Informationen liefern würden.

3 Stand der Technik

Im Folgenden soll der Stand der Technik bezüglich konditionaler Testausführung betrachtet werden. Dazu werden wir auf Erkenntnisse aus der Testfallreduktion und dem Regressionstesten eingehen, wobei wir bei letzterem auch Ideen aus der statischen Prozeßsteuerung begegnen werden.

3.1 Kosten-/Nutzen-Analyse von Techniken zur konditionalen Testausführung

In [MRE02] wird empirisch untersucht, welchen Kosten-/Nutzen-Verhältnis verschiedene Verfahren zur Testauswahl und –priorisierung beim Regressionstesten haben. Insbesondere werden die drei folgenden Verfahren betrachtet:

Testfall-Auswahl: Hier werden die Testfälle nach verschiedenen Kriterien selektiert, um nur die für den Regressionstest wirklich notwendigen Tests durchzuführen.

Testsuite-Reduktion: Bei diesem Ansatz werden redundante Testfälle in der Testsuite ermittelt, um die Anzahl der Testfälle zu verringern.

Testfall-Priorisierung: Hierbei werden die Testfälle, die zur Erreichung des Testziels nützlicher sind, vor den anderen Testfällen ausgeführt.

Da sich die Analyse im Artikel allerdings nur auf die Anwendung dieser Verfahren vor der Ausführung des ersten Testfalls beschränkt, sind die Erkenntnisse des Artikels für uns von geringerer Bedeutung. Allerdings erlauben uns die Begriffe eine Einordnung unseres Verfahrens: In Arbeitspaket 5.1, von dem dieser Bericht handelt, wird ein Verfahren zur Testfall-Auswahl mit Hilfe von Rückmeldungen aus dem Testprozess durchgeführt. Wie wir später sehen werden, erhalten wir auf dem gleichen Weg auch ein Verfahren zur Testsuite-Reduktion.

3.2 Ansätze aus der Testsuite-Reduktion

Wie oben beschrieben, versucht man bei der Testsuite-Reduktion, die Menge der Testfälle durch Entfernen von Redundanzen zu minimieren. Ein Teil dieses Prozesses besteht darin, Kriterien für die Bestimmung redundanter Testfälle zu entwickeln.

In [SSG+05] werden dazu drei Verfahren beschrieben, die alle aus einer gegebenen Menge von Testfällen eine möglichst kleine Testsuite für eine vorgeschriebene Menge von Anforderungen konstruieren. Allerdings benutzen alle

diese Verfahren nur a priori vorhandene Informationen, können also nicht direkt zur bedingten Ausführung herangezogen werden.

3.3 Ansätze aus dem Gebiet des Regressionstests

Als Regressionstest bezeichnet man die erneute Durchführung von Testfällen nach einer Anforderungs- oder Implementierungsänderung, um die neueste Version der Implementierung gegen die neueste Version der Anforderungen zu testen (vergleiche auch [Lig02]).

Damit nicht immer wieder die gesamte Testsuite ausgeführt werden muss, existieren in diesem Gebiet verschiedene Ansätze, wie man die benötigten Testfälle auswählen kann und welche Fälle nicht erneut überprüft werden müssen.

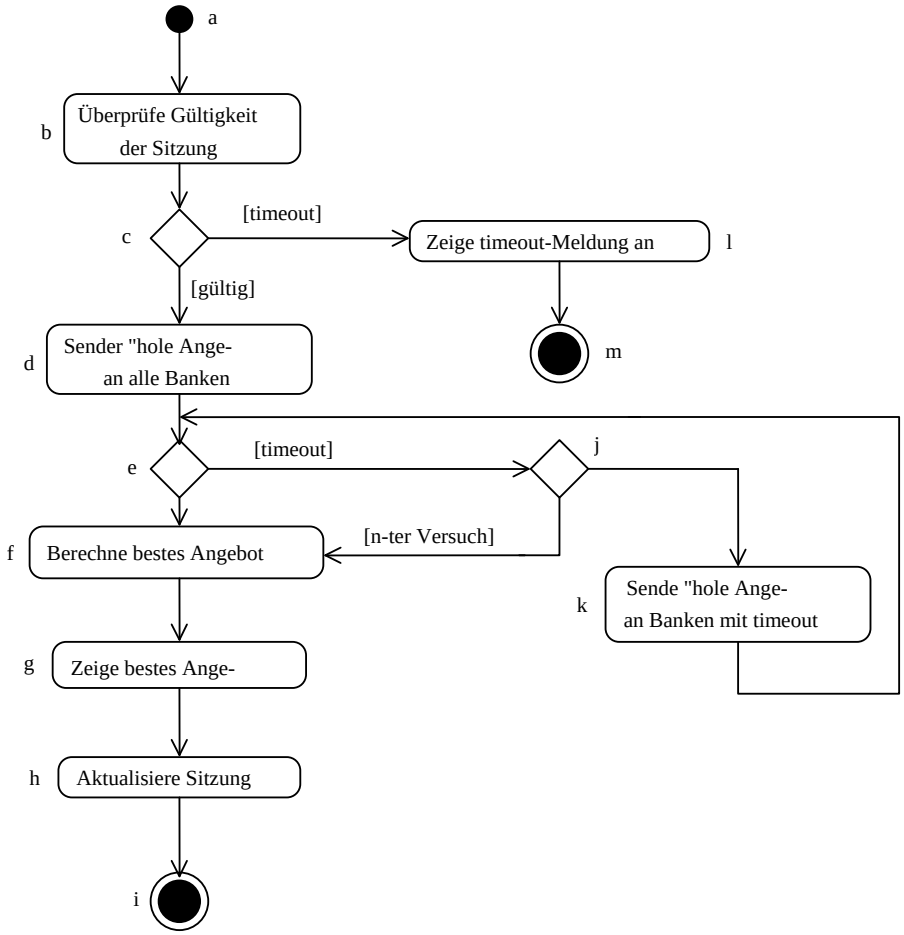
Das Ziel dieses Arbeitspaketes ist es, Kriterien zu finden, die sich hauptsächlich auf die bisher durchgeführten Testfälle beziehen. Aus diesem Grund sind Codebasierte Techniken, die den Löwenanteil der Literatur in diesem Gebiet ausmachen (vgl. [AHKL93], [CRV94], [LW89], [RH97]), in diesem Abschnitt als irrelevant anzusehen.

Ein Artikel, der sich in unsere Richtung bewegt, aber ein anderes Ziel als das dieses Arbeitspaketes verfolgt, ist [CPS02].

Es werden zwei Testauswahlkriterien vorgeschlagen, zum einen über Aktivitätsdiagramme oder ähnliche Verhaltensbeschreibungen des Systems, und zum anderen über eine Risikobewertung.

Betrachten wir zuerst das erste Kriterium. Wir zitieren das Beispiel aus dem Artikel. Dort sind folgendes Aktivitätsdiagram und folgende Zuordnung von Testfällen zu Zuständen des Diagrams angegeben:

Testfall	**Pfad**
t1	a,b,c,d,e,f,g,h,i
t2	a,b,c,d,e,j,k,e,f,g,h,i
t3	a,b,c,d,e,j,k,e,j,f,g,h,i
t4	a,b,c,l,m



Damit kann man allen Knoten und Kanten im Diagram Testfälle zuordnen:

Knoten	**Testfälle**	**Kante**	**Testfälle**
a	t1,t2,t3,t4	(a,b)	t1,t2,t3,t4
b	t1,t2,t3,t4	(b,c)	t1,t2,t3,t4
c	t1,t2,t3,t4	(c,d)	t1,t2,t3
d	t1,t2,t3	(c,l)	t4
e	t1,t2,t3	(d,e)	t1,t2,t3
f	t1,t2,t3	(e,f)	t1,t2

Knoten	Testfälle	Kante	Testfälle
g	t1,t2,t3	(e,j)	t2,t3
h	t1,t2,t3	(f,g)	t1,t2,t3
i	t1,t2,t3	(g,h)	t1,t2,t3
j	t2,t3	(h,i)	t1,t2,t3
k	t2,t3	(j,f)	t3
l	t4	(j,k)	t2,t3
m	t4	(k,e)	t2,t3
		(l,m)	t4

Diese Zuordnung erlaubt uns nun, genau die Testfälle auszuwählen, die von einer Programmänderung betroffen sind. Allerdings wird hierbei wiederum Wissen darüber vorausgesetzt, welche Teile im System betroffen sind, womit das Verfahren für Black-Box-Tests nicht geeignet ist.

Das zweite Kriterium funktioniert wie folgt: Jedem Testfall werden zwei Werte zugeordnet, die Fehlerwahrscheinlichkeit und die Risikobewertung des Testfalles. Dabei ist die Risikobewertung umso höher, je schwerwiegender ein Fehlschlagen des Testfalles ist. Als Priorisierungswerte verwendet man dann für jeden Testfall das Produkt von Fehlerwahrscheinlichkeit und Risikobewertung und führt die Testfälle sortiert nach Priorisierungswert durch. Allerdings gehört dieses Kriterium eher in den Bereich der Testfall-Priorisierung als in den Bereich der bedingten Ausführung.

Ein weiterer interessanter Artikel ist [KP02]: Dort wird für die Auswahl von Testfällen vorgeschlagen, jedem Testfall eine Wahrscheinlichkeit zuzuordnen und dann so lange zufällig Testfälle aus der Testsuite zu wählen, bis die Testressourcen verbraucht sind.

Für die Ausführung eines Testfalls wird ein Qualitätsmaß festgelegt, zum Beispiel ob der Testfall einen Fehler erkannt hat. Weiterhin wird für jeden Testfall ein Qualitätswert gespeichert, der das bisherige Verhalten des Tests bewertet, und der proportional zur Auswahlwahrscheinlichkeit ist. Dieser Wert berechnet sich wie folgt: Sei Q die Qualität einer Ausführung, H der bisherige Qualitätswert, H' der neue Qualitätswert und a ein festgelegter Parameter. Dann gilt $H' = aQ + (1-a)H$.

Dieser Ansatz ist in so weit nützlich, als das bei bekannten Qualitätswerten eine Auswahl getroffen werden kann. Die Ergebnisse der anderen Testfälle, die beim aktuellen Testlauf bereits durchgeführt wurden, gehen nicht ein.

4 Abhängigkeitsanalyse

4.1 Bewertungskriterien der Testergebnisse

Je nach Art der Testfälle ist es möglich, dass es nicht nur zwei mögliche Testausgänge (bestanden oder durchgefallen) gibt, sondern auch weitere Ausgänge. Insbesondere bei Tests auf nicht funktionale Eigenschaften sind auch Ergebnisse wie „knapp durchgefallen" und „knapp bestanden" bei Timing-Tests oder „Fehlschlag wegen Umgebungsproblem" denkbar.

Um ein allgemeines Verfahren entwerfen zu können, werden wir im Folgenden die möglichen Testausgänge in drei Klassen einteilen:

1. Erfolgreich,
2. Erfolglos ohne Einfluss auf andere Testfälle,
3. Erfolglos mit Einfluss auf andere Testfälle.

4.2 Abhängigkeiten

Wie zuvor in Kapitel 3 beschrieben, gibt es zwei mögliche Abhängigkeiten, die für die Selektion der Testfälle von Bedeutung sind:

- Abhängigkeit zwischen Testfällen
- Abhängigkeit zwischen Testergebnissen und weiteren Testfällen

Der Fokus von D5.1 liegt besonders auf der Erforschung der Abhängigkeiten zwischen Testergebnissen und weiteren Testfällen.

Für die weitere Diskussion machen wir vorerst zwei Annahmen:

1. Alle Fehler treten reproduzierbar auf. Insbesondere wird jede Eingabefolge, die einmal zu einem Fehler führt, immer zu einem Fehler führen.
2. Wird bei einer Testfallausführung ein Fehler erkannt, so kann eindeutig bestimmt werden, in welchem Schritt der Fehler auftrat.

Zusammengenommen bedeutet das: Wenn bei der Ausführung eines Testfalls ein Fehler auftritt, so können wir ein Präfix des Testfalls ermitteln, das bei jeder

Ausführung zu einer Fehleraufdeckung führt. Solch ein Präfix nennen wir Fehlerzeuge.

Formal beschreibt man diese Eigenschaft wie folgt: Betrachte zwei Testfälle t und t'. Wir bezeichnen mit t(i) bzw. t'(i) die i-te Testeingabe, die vom entsprechenden Test gemacht wird. Angenommen, es gibt ein k, so dass für alle $i < k$ gilt: $t(i) = t'(i)$. Tritt dann bei der Ausführung von Testfall t ein Fehler in weniger als k Schritten auf, sagen wir in i Schritten, so wird dieser Fehler auch bei der Ausführung von t' auftreten, sofern es sich um einen reproduzierbaren Fehler handelt. Wir nennen dann die Folge t(0) t(1) … t(i) einen Fehlerzeugen.

Ausgehend davon definieren wir also ein Kriterium für die bedingte Ausführung von Testfällen wie folgt: Ein Testfall wird nur dann ausgeführt, wenn es keinen Fehlerzeugen gibt, der Präfix der Eingaben des Tests ist. Weiterhin können wir ein zusätzliches Kriterium angeben, wann Testfälle redundant sind: Ist die Eingabefolge eines Testfalls t Präfix der Eingabefolge eines Testfalls t', so ist t redundant, denn er wird mitausgeführt, wenn t' ausgeführt wird.

Ein Algorithmus zum effizienten Überprüfen dieser Kriterien wird im folgenden Kapitel angegeben.

5 Algorithmen zur konditionalen Testfallreduktion

Wie im letzten Kapitel beschrieben, brauchen wir einen effizienten Algorithmus, um zu prüfen, ob ein Präfix einer Testfall-Eingabe ein Fehlerzeuge ist.

Unser Ansatz wird auf der Konstruktion eines Präfix-Baums beruhen, in dem die Eingabesequenzen der Testfälle sowie die Fehlerzeugen abgelegt werden. Der Aufbau dieses Baumes wird in Abschnitt 5.1 beschrieben, seine Erzeugung und Benutzen bei der Testdurchführung in den folgenden Kapiteln.

5.1 Der Präfix-Baum

Zunächst wollen wir formal den Begriff des Präfixbaums definieren. Sei □ ein Alphabet und T ein Baum, dessen Kanten mit Elementen von □ beschriftet sind. Dann nennen wir T einen Präfix-Baum für eine Menge W von Worten aus □*, wenn die folgenden drei Eigenschaften erfüllt sind:

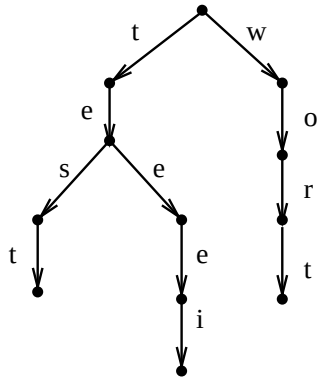
1. Für jeden Knoten v in T und jeden Buchstaben s in □ gibt es höchstens eine ausgehende Kante von v, die mit s beschriftet ist.

2. Für jedes Wort w in W gibt es einen Pfad von der Wurzel durch T, so dass die Buchstaben der Kantenbeschriftungen w bilden.

3. Für jedes Blatt von T gibt es ein Wort w, dessen Buchstaben am Pfad von der Wurzel zum Blatt stehen.

Die dritte Bedingung dient dabei dazu, Pfade im Baum zu verhinden, die zu keinem Wort in w gehören. Ohne diese Bedingung wäre es zum Beispiel möglich, zu der Wortmenge {x} einen Baum zu konstruieren, der neben einer Kante von der Wurzel mit Beschriftung x eine beliebige Menge weiterer Kanten enthält und trotzdem die beiden anderen Bedingungen erfüllt.

Die folgende Abbildung zeigt ein Beispiel eines solchen Baums:



Dieser Baum ist ein Präfix-Baum für die Wortmenge {test, teeei, wort}, aber auch für die Wortmenge {test, tee, teeei, wort}. Der Baum ist kein Präfix-Baum von {test, teeei}, da es eine Kantenfolge von Wurzel zu Blatt gibt, die mit „wort" beschriftet ist. Auch ist er kein Präfix-Baum von {test,teeei,wort,wald}, da „wald" keinem Pfad durch den Baum entspricht.

Wenn wir nun als Alphabet die Menge der Testeingaben nehmen und einen Präfix-Baum T für die Eingaben der Testfälle konstruieren, so können wir diesen benutzen, um alle Testfälle zu ermitteln, die aufgrund eines aufgetretenen Fehlers nicht mehr ausgeführt werden müssen: Sei z der Fehlerzeuge. Man folge dem zu z korrespondierenden Pfad durch T und merkt sich den Teilbaum, der am Ende dieses Pfades hängt. Falls ein Test zur Ausführung ansteht, dessen Testeingabe-Folge in diesem Teilbaum endet, wird dieser Test aufgrund des vorher erkannten Fehlers fehlschlagen und braucht nicht ausgeführt zu werden.

Von jetzt an werden wir weiterhin auch erlauben, dass ein fehlgeschlagener Testfall die Ausführung weiterer Testfälle mit gemeinsamen Präfix nicht blockiert. Wir sprechen dabei von erfolglosen Testfällen; denkbar wäre beispielsweise ein Testfall, bei dem transiente Fehler aufgetreten sind.

Zur weiteren Effizienzsteigerung führen wir eine Markierung der Knoten des Baumes ein. Zum einen betrachten wir dazu das Wort w, das vom Pfad von der Wurzel zu einem Knoten k induziert wird. Ist w die Folge von Testeingaben eines Testfalles, so wird in k ein Verweis auf diesen Testfall untergebracht. Wenn dieser Testfall erfolgreich ausgeführt wurde, wird auch das im Knoten vermerkt. Ist w ein Fehlerzeuge, so wird k als „Fehlerknoten" markiert.

Die folgende Tabelle enthält kurze Bezeichnungen für die auftretenden Knotenarten und ihre graphische Darstellung. Jede Sorte Knoten wird durch eine Form dargestellt.

Knoten korrespondiert zu	Bezeichnung	Form
nicht ausgeführtem Testfall	Offen	Quadrat
erfolgreichem Testfall	Erfolgreich	gedrehtes Quadrat
erfolglosem Testfall	Erfolglos	Dreieck
Fehlerzeuge	Fehlerhaft	Raute
keinem Testfall oder Zeugen	Normal	Kreis

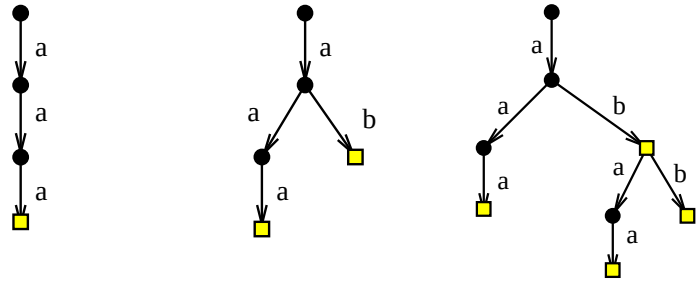
5.2 Algorithmen

5.2.1 Konstruktion des Präfix-Baums

Der Präfixbaum wird iterativ konstruiert, indem man nacheinander alle Testeingaben in den Präfixbaum einfügt. Dabei wird so lange wie möglich bereits vorhandenen Kanten gefolgt. Die Knoten, die an den Enden der Testsequenzen stehen, werden mit dem entsprechenden Verweis versehen und als „noch nicht ausgeführter Testfall“ markiert.

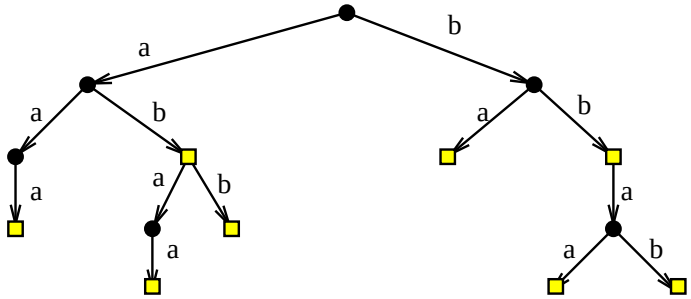
Man beachte, dass auch innere Knoten Testfälle darstellen können. Das tritt beispielsweise auf, wenn man die Eingabefolgen „abc“ und „abcabc“ betrachtet. Wie oben bemerkt, kann man allerdings Testfälle, die zu inneren Knoten korrespondieren, als redundant aus der Testsuite entfernen. Alternativ kann man diese Testfälle auch bei der Ausführung überspringen

Als Beispiel für die Ausführung dieses Algorithmus betrachten wir die Testfall-Eingabesequenzen (kurz Testsequenzen) aaa, ab, abaa, abb, ba, bb, bbaa und bbab. Wir zeigen zuerst den Baum nach dem Einfügen der ersten, zweiten und vierten Sequenz:



Erster Schritt Zweiter Schritt Vierter Schritt

Nach allen Einfügeschritten erhält man schließlich den folgenden Präfixbaum:



5.2.2 Bedingte Testfallausführung: automatische Testfall-Auswahl ohne Priorisierung

Im Folgenden soll ein Algorithmus angegeben werden, der die vorhandenen Testfälle in längenlexikographischer Reihenfolge durchläuft und nur solche Testfälle ausführt, die nach unserem Kriterium nicht auf Grund bisherigen Wissens redundant sind.

Als Strategie verwenden wir eine Breitensuche auf unserem Präfixbaum: Wir durchlaufen vom Wurzelknoten aus alle Niveaus nacheinander, lassen allerdings alle Teilbäume mit fehlerhaften Wurzeln weg. Stoßen wir dabei auf einen offenen Knoten, wird der entsprechende Testfall ausgeführt.

Der folgende Algorithmus implementiert diese Strategie für einen Präfixbaum T:

```
Q := leere Warteschlange
hänge die Wurzel von T an Q an.
while Q nicht leer do
    comment Schleifenanfang
    k := erstes Element von Q
    entferne k aus Q
    if k normal, erfolglos oder erfolgreich then
```

füge alle Söhne von k zu Q hinzu
else if k erfolgreicher Testfall **then**
if k hat Nachfolger **then**
comment Testfall redundant
füge alle Söhne von k zu Q hinzu
else
führe zu k gehörenden Testfall durch.
comment Testfall fertig
if Testfall bis zum Ende ausgeführt **then**
markiere alle offenen Knoten auf dem Pfad zu
k als erfolgreich bzw. erfolglos, inklusive k.
elif Testfall bei Knoten f abgebrochen **then**
markiere offene Knoten auf dem Pfad zu f
als erfolgreich bzw. erfloglos, exklusive f.
markiere f als fehlerhaft
entferne alle Knoten aus Q, bei denen f
auf dem Pfad zur Wurzel liegt.
fi
comment Markierung fertig
fi
fi
od

Als Beispiel für die Ausführung des Algorithmus' betrachten wir einen Testdurchlauf auf dem Präfixbaum des vorigen Abschnittes, dessen Blätter wir hier zur Verdeutlichung durchnummeriert haben:

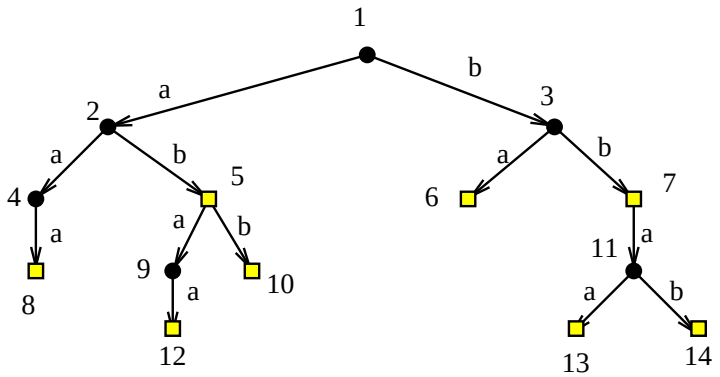


Abbildung 1: Vor der Testfallausführung

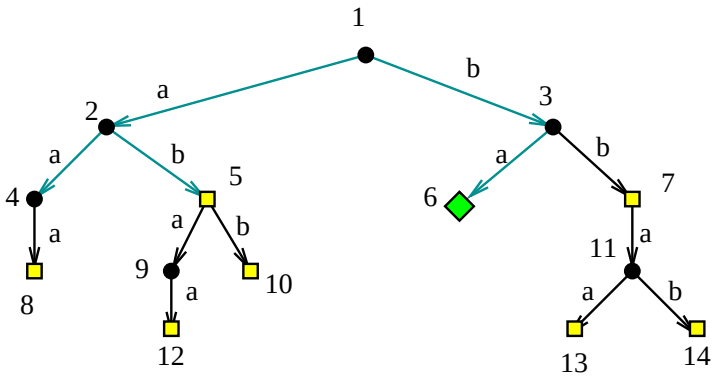


Abbildung 2: Nach der ersten Testfallausführung

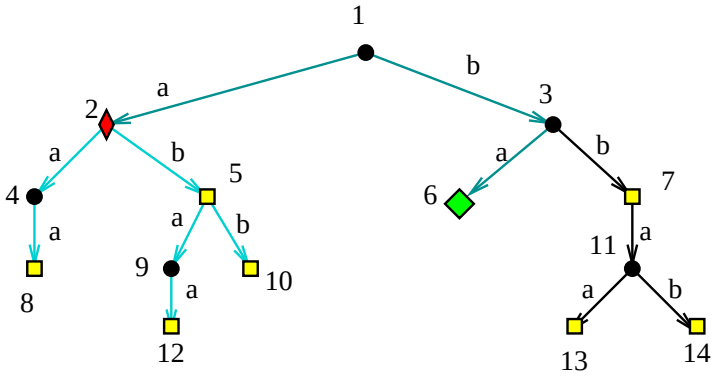


Abbildung 3: Nach der zweiten Testfallausführung

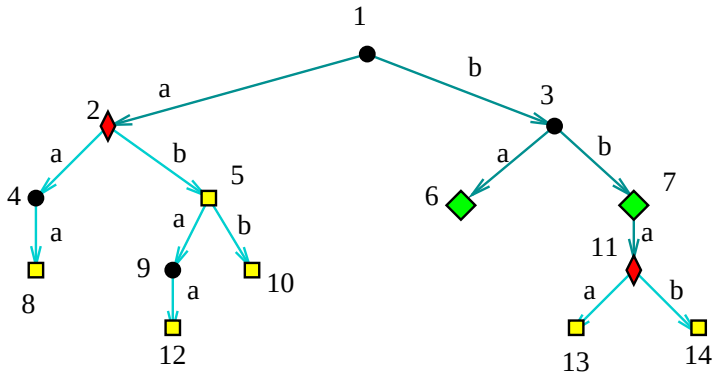


Abbildung 4: Nach der letzten Testfallausführung

Wir durchlaufen nun diesen Baum mit dem Algorithmus von oben. Zwischenstände des Durchlaufs sind in der folgenden Tabelle dargestellt:

Programm-position	**K**	**Q**	**Was geschieht?**
Schleifenanfang	1	[]	Normaler Knoten, nur dessen Kinder anfügen. Danach ist Q=[2,3].
Schleifenanfang	2	[3]	Normaler Knoten, nur Kinder anfügen. Danach ist Q=[3,4,5].
Schleifenanfang	3	[4,5]	Normaler Knoten.
Schleifenanfang	4	[5,6,7]	Normaler Knoten
Schleifenanfang	5	[6,7,8]	Ein offener Knoten ist erreicht. Der Testfall ist allerdings redundant und wird deswegen nicht ausgeführt.
Schleifenanfang	6	[7,8,9,10]	Ein offener, nicht redundanter Knoten ist erreicht. Der Testfall wird erfolgreich ausgeführt.
Testfall fertig	6	[7,8,9,10]	Knoten 6 wird als erfolgreich markiert.
Markierung fertig	6	[7,8,9,10]	Baumstruktur: s. Abbildung 2
Schleifenanfang	7	[8,9,10]	Ein offener, redundanter Knoten.
Schleifenanfang	8	[9,10,11]	Ein offener, nicht redundanter Knoten ist erreicht. Wir nehmen an, dass der Testfall mit Fehlerzeuge a abbricht.
Testfall fertig	8	[9,10,11]	Aufgrund des gefundenen Fehlerzeugen wird Knoten 2 als fehlerhaft markiert und der Teilbaum unter 2 aus Q entfernt.
Markierung fertig	8	[11]	Baumstruktur: s. Abbildung 3
Schleifenanfang	11	[]	Normaler Knoten
Schleifenanfang	13	[14]	Ein offener, nicht redundanter Knoten ist erreicht. Wir nehmen an, dass der Testfall mit Fehlerzeuge bba ab-

Programm-position	K	Q	Was geschieht?
			bricht.
Testfall fertig	13	[14]	Aufgrund des gefundenen Fehlerzeugen wird Knoten 11 als fehlerhaft markiert und der Teilbaum unter 11 aus Q entfernt. Da bb erfolgreich ausgeführt wurde, wird weiterhin Knoten 7 als erfolgreich markiert.
Markierung fertig	13	[]	Baumstruktur: s. Abbildung 4

Somit haben wir einen Durchlauf aller potentiell erfolgreichen Testfälle durchgeführt. Wir haben von acht möglichen Testfällen nur drei durchgeführt, denn die anderen fünf wurden während der Testsuite-Ausführung weggelassen, da sie keine neue Information liefern würden. Das ist auch das Maximum, was wir in diesem Fall mit unseren Redundanzkriterien erreichen können.

5.2.3 Wiederholung der Testfallausführung nach Fehlerkorrektur

Wenn der Präfixbaum einer Menge von Tests mit Markierungen erhalten bleibt, kann man ihn auch für die Testselektion von Regressionstests mitverwenden. Dazu kann man wie folgt vorgehen:

- Man entfernt die Fehler-Markierung von den Knoten des Baumes, von denen man ausgeht, dass der diese Markierung verursachende Fehler beseitigt wurde. Auf diese Art erhält man eine Menge von offenen Knoten.

- Aus den erfolgreichen Knoten wählt man diejenigen aus, die in diesem Testdurchlauf erneut getestet werden müssen. Hierzu benutzt man die üblichen Testauswahlstrategien des Regressionstests.

- Sollen neue Testfälle hinzugefügt werden, benutzt man die gleiche Methode wie beim Aufbau des Präfixbaums.

Danach kann man die Testfallausführung genau wie oben beschrieben durchführen.

5.2.4 Mögliche Erweiterungen

Wenn es möglich ist, den Zustand des Testsystems zu gegebenen Zeitpunkten einzufrieren, so kann man eine weitere Optimierung vornehmen: An Verzweigungspunkten des Baumes kann der Systemzustand nach Teilausführung eines Testfalls abgespeichert werden. Alle weiteren Testfälle, die über diesen Verzweigungspunkt laufen, können dann vom eingefrorenen Systemzustand aus durchgeführt werden, wodurch die Durchführung doppelter Schritte vermieden werden kann.

6 Bewertung

Der in diesem Ergebnisbericht vorgestellte Ansatz basiert auf folgenden Annahmen:

1. Alle Fehler treten reproduzierbar auf. Insbesondere wird jede Eingabefolge, die einmal zu einem Fehler führt, immer zu einem Fehler führen.

2. Wird bei einer Testfallausführung ein Fehler erkannt, so kann eindeutig bestimmt werden, in welchem Schritt der Fehler auftrat.

Dann ermöglichen die oben angegebenen Algorithmen, die Ausführung von Testfällen zu unterbinden, die nicht in der Lage sind, neue Informationen zu liefern. Gleichzeitig ermöglicht uns das Verfahren, sehr einfach Regressionstests durchzuführen. Ein dritter Vorteil ist, dass beim Einfügen redundante Tests erkannt werden können: Im Beispiel aus Kapitel 5 wird man beim Erstellen des Präfixbaums feststellen, dass der Testfall „bb" vom Testfall „bbaaa" subsumiert wird. Diese Feststellung kann zur Testsuite-Reduktion verwendet werden.

Es bleibt die Frage nach der Effizienz des Verfahrens. Dazu kann folgende Überlegung genutzt werden: Angenommen, jeder Testfall der Testmenge müsste tatsächlich ausgeführt werden (schlimmster Fall), dann können wir die zusätzlichen Laufzeitkosten für die Testanwendung wie folgt abschätzen:

Zusätzlich zu sowieso notwendigen Einlesen der Testfälle und Ausführen der einzelnen Schritte ist es dann notwendig, für jeden Testschritt eine Kante in den Präfixbaum einzufügen und diese später zu traversieren. Bezeichnen wir die Kosten für das Einfügen und Traversieren einer Kante mit *b* und die Kosten für das Einlesen und Ausführen eines Schrittes mit *a*. Gibt es insgesamt *n* auszuführende Schritte in allen Tests, so wären die Kosten des Verfahrens in diesem Fall $(a+b)n$, während die Ausführung ohne diesen Algorithmus an Zeitschritte kostet.

Nehmen wir nun weiterhin an, dass durch dieses Verfahren *m* Testschritte eingespart werden. Damit reduzieren sich die Kosten auf $bn + a(n-m)$. Durch Vergleich mit den Kosten ohne Reduktion erhält man, dass die Testzeit dann verringert wird, wenn $an - (bn + a(n-m)) > 0$ gilt, also die Bedingung $\frac{b}{a} n < m$ erfüllt ist.

Da die Operationen „Testschritt in den Baum eintragen" und „Traversieren einer Baumkante" jeweils sehr einfach sind und im Wesentlichen aus dem Zugriff

auf einige wenige Hauptspeicherpositionen bestehen, können wir annehmen, dass *b* einen kleinen, konstanten Wert hat.

Umgekehrt können wir annehmen, dass die Operationen „Testschritt einlesen" und „Testschritt ausführen" im Allgemeinen zumindest die Ausführung größerer Programmteile anstoßen, zum Beispiel Einlesen und Parsen eines Testfalls oder Ausführung eines Programmteils des zu testenden System. Damit wird *a* im Allgemeinen deutlich größer als *b* sein.

Nehmen wir beispielsweise an, dass die Zeitkosten für die Ausführung eines Testschrittes im Durchschnitt mindestens das hundertfache der Eintragung und Traversierung einer Kante im Präfixbaum betragen. Diese Annahme ist beispielsweise dann gerechtfertigt, wenn das System regelmäßig größere Berechnungen durchführt oder mit Peripheriegeräten oder anderen Systemen kommuniziert. Dann gilt für den Quotienten von *a* und *b*, dass $a/b < 1\%$ ist. Somit würde sich in diesem Beispiel die Anwendung des Verfahrens lohnen, wenn 1% alle Testschritte eingespart werden könnte. Wäre die durchschnittliche Länge eines Testfalles 5 Schritte, so könnte man durch Einsparung von 0,2% der Testfälle im Allgemeinen schon die Kosten des Verfahrens amortisieren.

7 Zusammenfassung

In diesem Bericht wird ein Verfahren entworfen, um bei der Ausführung einer Menge von Testfällen während des Testprozesses solche Testfälle zu erkennen, deren Ausführung keine neuen Informationen liefern würde. Insbesondere werden solche Testfälle erkannt, deren Ausgang durch das Ergebnis bereits durchgeführter Tests schon feststeht.

Dazu werden zunächst in Kapitel 3 bereits bekannte Techniken untersucht, wobei hier insbesondere Ergebnisse aus dem Bereich des Regressionstestens betrachtet werden. Anschließend werden in Kapitel 4 die möglichen Abhängigkeiten zwischen Testfällen untereinander und zu bereits ermittelten Testergebnissen diskutiert. Darauf aufbauend wird in Kapitel 5 eine Datenstruktur zur Erkennung solcher Abhängigkeiten entworfen, die als Präfixbaum bezeichnet wird. Weiterhin werden Algorithmen entwickelt, um Präfixbäume zu errechnen und eine Menge von Testfällen so auszuwerten, dass der zugehörige Präfixbaum dazu benutzt werden kann, mit einer möglichst geringen Zahl von ausgeführten Testfällen alle Informationen, die aus der Testfallmenge gewonnen werden können, ermittelt werden. Schließlich wird der Algorithmus in Kapitel 6 bewertet, wobei wir feststellen, dass die Ausführung des Algorithmus nur wenige zusätzliche Komplexität einführt, aber in der Lage ist, große Einsparungen an auszuführenden Tests zu ermöglichen.

Für die folgenden Arbeitspakete liefert dieser Algorithmus einen Baustein, der bei der dynamischen Priorisierung zusammen mit weiteren Methoden ein effizientes Verfahren ermöglicht.

Literatur

[AHKL93] H. Agrawal, J.R. Horgan, E.W. Krauser, and S.A. London. Incremental regression testing. *Software Maintenance ,1993. CSM-93, Proceedings., Conference on*, pages 348–357, Sep 1993.

[CPS02] Yanping Chen, Robert L. Probert, and D. Paul Sims. Specification-based regression test selection with risk analysis. In *CASCON '02: Proceedings of the 2002 conference of the Centre for Advanced Studies on Collaborative research*, page 1. IBM Press, 2002.

[CRV94] Yih-Farn Chen, David S. Rosenblum, and Kiem-Phong Vo. Testtube: a system for selective regression testing. In *ICSE '94: Proceedings of the 16th international conference on Software engineering*, pages 211–220, Los Alamitos, CA, USA, 1994. IEEE Computer Society Press.

[KP02] Jung-Min Kim and Adam Porter. A history-based test prioritization technique for regression testing in resource constrained environments. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 119–129, New York, NY, USA, 2002. ACM.

[Lig02] Peter Liggesmeyer. *Software-Qualität*. Spektrum Akademischer Verlag, 2002.

[LW89] H.K.N. Leung and L. White. Insights into regression testing [software testing]. *Software Maintenance, 1989., Proceedings., Conference on*, pages 60–69, Oct 1989.

[MRE02] A.G. Malishevsky, G. Rothermel, and S. Elbaum. Modeling the cost-benefits tradeoffs for regression testing techniques. *Software Maintenance, 2002. Proceedings. International Conference on*, pages 204–213, 2002.

[RH97] Gregg Rothermel and Mary Jean Harrold. A safe, efficient regression test selection technique. *ACM Trans. Softw. Eng. Methodol.*, 6(2):173–210, 1997.

[SSG+05] S. Sprenkle, Sreedevi Sampath, E. Gibson, L. Pollock, and A. Souter. An empirical comparison of test suite reduction techniques for user-session-based testing of web applications. *Software Maintenance, 2005. ICSM'05. Proceedings of the 21st IEEE International Conference on*, pages 587–596, Sept. 2005.

Dokumenten Information

Titel:	D-5.1: Ansatz für konditionale Testfallreduktion basierend auf vorausgegangenen Testergebnissen
Datum:	Dezember 2008
Report:	IESE-044.09/D
Status:	Final
Klassifikation:	Öffentlich