Lean Testing

Prof. Dr. Andreas Spillner (i.R.)
Andreas.Spillner@hs-bremen.de
Prof. Dr. Ulrich Breymann (i.R.)
Urlich.Breymann@hs-bremen.de

Abstract
Es ist das Ziel eines jeden Softwareentwicklers[1], Programme mit möglichst wenigen Fehlern zu schreiben. Wie man weiß, ist das weiter gehende Ziel einer fehlerfreien Software nicht zu erreichen, von sehr kleinen Programmen abgesehen. Aber: Wie prüfe ich mein Programm(teil) auf Fehler und wie groß darf ein vertretbarer Testaufwand sein? Dieser Beitrag versucht, anhand eines einfachen Beispiels zu zeigen, was »Lean Testing« ist und was es leisten kann.

Stichworte
Testentwurfsverfahren, Klassifikationsbaum-Methode, Kombinatorisches Testen, Programmierung

1. Fehlerreduktion zu angemessenen Kosten

Es ist möglich, die Anzahl der Fehler zu reduzieren. Dabei helfen erstens konstruktive Maßnahmen wie die Einhaltung von Programmierrichtlinien und das Schreiben eines verständlichen, wartungsfreundlichen Programmtextes (Stichwort Clean Code). Zweitens hilft das Testen, also die Prüfung der Software, ob sie den Anforderungen genügt und ob sie Fehler enthält.

Wie viel Aufwand soll in den Test gesteckt werden? Einerseits möglichst wenig, um die Kosten niedrig zu halten, andererseits möglichst viel, um dem Ziel der Fehlerfreiheit nahezukommen. Letztlich geht es darum, einen vernünftigen Kompromiss zwischen diesen beiden Extremen zu finden.

Der aus dem agilen Umfeld bekannte Begriff »lean« bedeutet beim Testen, sich auf das Wichtige zu konzentrieren, um diesen Kompromiss zu erreichen. Die Frage des Aufwands ist aber nur vordergründig ausschließlich für Tester von Bedeutung - sie betrifft auch Softwareentwickler, da sie auch testen.

»Lean Testing« [1] steht für einen Ansatz, der auf der einen Seite alle wichtigen Testfälle zur Prüfung der Software berücksichtigt, auf der anderen Seite aber den Testaufwand in einem überschaubaren Rahmen hält.

2. Brücke zwischen Programmierung und Test

Tatsächlich checkt ein Softwareentwickler seinen Code erst ein, wenn er ihn auf seiner Ebene, also der Ebene der Komponente oder Unit, getestet hat. Er ist interessiert an der Ablieferung guter Software und an der Anerkennung dafür. Er muss aber auch darauf achten, nicht mehr Zeit als angemessen zu investieren. Die aktuelle Entwicklung, bei der Softwareerstellung keine strikte (personenbezogene) Trennung zwischen Implementierung und Test auf Unit-Ebene vorzusehen (z.B. TDD und Test-first-Ansatz), erfordert eine Brücke zwischen Programmierung und Testen für den Entwickler. Ihm soll bekannt sein, welche Testentwurfsverfahren es gibt und wie sie mit vertretbarem Aufwand auf seiner Ebene eingesetzt werden können.

3. Reduktion der Anzahl der Testfälle am Beispiel

Zwei Ansätze zum »Lean Testing« zeigen, wie die Anzahl der Testfälle reduziert werden kann. Sie werden am folgenden Beispiel erörtert: Ein Sportverein bietet die Sportarten Tischtennis, Turnen, Volleyball, Basketball, Handball und Fitnesstraining an. Jede Sportart ist aus organisatorischen Gründen einer Abteilung zugeordnet, wobei die Sportarten Volleyball, Basketball und Handball zu einer Abteilung Ballsport zusammengefasst werden. Damit wird auch den verschiedenen Kosten der Sportarten Rechnung getragen.

Der monatliche Mitgliedsbeitrag setzt sich zusammen aus einem Grundbetrag und einem Betrag für die in Anspruch genommene Abteilung (mindestens eine). Dabei kann mehr als eine Abteilung gewählt werden. Die monatlichen Zusatzbeiträge für die Abteilungen sind: Tischtennis 5, Turnen 11, Ballsport 9, Fitnesstraining 10 Euro. Die zu

[1] Geschlechtsbezogene Formen meinen im gesamten Text Frauen, Männer und alle anderen.

testende Anwendung soll den monatlichen Mitgliedsbeitrag berechnen. Bei dem Grundbeitrag der Mitgliedsgebühr werden verschiedene Kategorien (Mitgliedsstatus) unterschieden: Der monatliche Grundbeitrag beträgt für

- Kind oder Jugendlicher 7 Euro,
- Erwachsener 18 Euro,
- Studierende, Rentner, Sozialhilfeempfänger (Ermäßigter Beitrag) 8 Euro
- passive Mitglieder 20 Euro.

Ein passives Mitglied fördert den Verein durch seine Mitgliedschaft, ohne aktiv in einer der Abteilungen mitzuwirken.

Wir haben somit vier Abteilungen für die Sportarten und vier Status der Mitgliedschaft, wobei die Abteilungen mehrfach gewählt werden können. Rein mathematisch ergeben sich somit aus den 16 möglichen Kombinationen von Sportarten kombiniert mit 4 Status 64 mögliche Testfälle. Sind alle zu testen oder reicht eine Auswahl an Testfällen aus? Wenn ja, welche? Sehen wir uns die zwei Ansätze genauer an.

3.1 Lean Testing mit dem Klassifikationsbaum

Um bei den zu erstellenden Testfällen leicht entscheiden zu können, welche Sportarten mit welchem Mitgliedsstatus kombiniert werden sollen, wird die Klassifikationsbaum-Methode verwendet. Mit der Klassifikationsbaum-Methode kann festgelegt werden, welche Kombinationen wie intensiv zu testen sind. Die grafische Darstellung bietet dabei einen sehr guten Überblick über die gewählten Testfälle.

Der Eingabedatenraum des Testobjekts wird bei dieser Methode nach Aspekten strukturiert, die als relevant eingeschätzt werden. Ein Aspekt oder Gesichtspunkt wird als Klassifikation bezeichnet. Auf der konkreten Ebene der Testfälle entsprechen die Klassifikationen den Parametern eines Testfalls und die Werte – oder genauer die Menge von Werten –, die ein Parameter annehmen kann, heißen in der Terminologie der Klassifikationsbaum-Methode Klassen. Zur besseren Strukturierung können Klassifikationen zu Kompositionen zusammengefasst werden. In unserem Beispiel werden die einzelnen Abteilungen als relevant für den Test angesehen und bilden jeweils eine Klassifikation. Der Oberbegriff »Abteilungen« kann als strukturierende Komposition gesehen werden (s. Abb. 1).

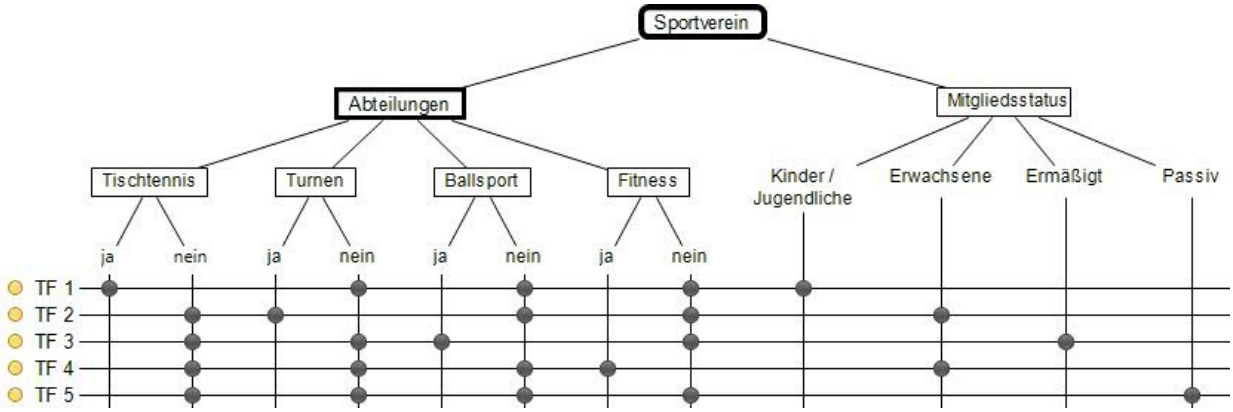

Abb. 1: Klassifikationsbaum mit fünf Testfällen

Zu jeder Klassifikation gehören zwei Klassen (Werte), nämlich »ja« und »nein«. Die Werte geben für ein bestimmtes Mitglied an, ob es in der betreffenden Abteilung aktiv ist. Die zum Mitgliedsstatus gehörenden vier Klassen (Werte) sind Kind/Jugendlicher, Erwachsener, Ermäßigt, Passiv. Ein bestimmtes Mitglied kann nur zu einer der vier Klassen gehören.

Was ist nun zu tun? Die Klassen der unterschiedlichen Abteilungen sind mit den unterschiedlichen Mitgliedsstatus zu kombinieren und entsprechende Testfälle daraus abzuleiten. Das heißt, durch die Kombination der Klassen unterschiedlicher Klassifikationen werden Testfälle definiert. Die fünf Testfälle (TF1-TF5) in Abbildung 1 wurden so gewählt, dass jede Klasse wenigstens einmal vorkommt. Dies wäre schon ein Minimalkriterium für den Test.

3.1.1 Systematische Auswahl der Testfälle

Welche weiteren Überlegungen sind nun anzustellen, um mit möglichst wenig Testfällen die Anwendung als ausreichend - lean - getestet anzusehen?

Zuerst ist der Sonderfall »passives Mitglied« zu untersuchen. Hier bestehen Abhängigkeiten zu den anderen Klassen, da ein passives Mitglied keiner Abteilung zugeordnet werden kann. Damit haben wir schon einen wichtigen Testfall (TF5 in

Abbildung 1). Sind Testfallkombinationen mit dem Mitgliedsstatus Passiv und einer (oder mehreren) Zuordnungen zu den Abteilungen erforderlich? Erwartet wird eine Fehlermeldung, da diese Kombination nicht den Anforderungen entspricht. Ein Testfall zur Prüfung der Ablehnung einer solchen fehlerhaften Kombination ist sinnvoll, sofern er überhaupt durchführbar - also über die Benutzungsschnittstelle auswählbar – ist.

Drei unterschiedliche Mitgliedsstatus und vier Abteilungen, die untereinander beliebig kombiniert werden können, sind nun zu untersuchen. Aus den insgesamt 16 Möglichkeiten für die Belegung der Abteilungen kombiniert mit den 3 Status ergeben sich 48 Testfälle. In den 16 Kombinationen der Abteilungen ist auch die Kombination enthalten, die keine Sportart auswählt (viermal »nein«). Dies ist wieder ein Sonderfall, denn diese Kombination entspricht dem Status Passiv. Es bleiben somit 15·3 = 45 Kombinationen von Abteilungszugehörigkeiten mit aktiven Mitgliedschaften – eine (zu) hohe Anzahl von Testfällen für »Lean Testing«.

Was muss eigentlich geprüft werden, was ist der »Kern« der Anwendung? Der Mitgliedsbeitrag soll ermittelt werden, also konzentrieren wir uns bei unseren Testüberlegungen auf diesen Aspekt. Der Beitrag ist abhängig vom Status und von der Wahl der Abteilungen.

Betrachten wir hierzu die Testfälle aus Abb. 1 (bzw. Abb. 2): Die Testfälle TF1- TF4 prüfen jeweils eine Zugehörigkeit zu einer Abteilung und kombinieren diese relativ willkürlich mit einem Mitgliedsstatus, allerdings so, dass jeder Status in einem Testfall vorkommt. Testfall 5 ist der oben bereits diskutierte Sonderfall. Die Testfälle prüfen in keinem Fall eine Zugehörigkeit zu mehreren Abteilungen.

3.1.2 45 Testfälle?

Müssen nun doch alle 45 Kombinationen geprüft werden? Nein, bei dieser Anwendung halten wir folgende Testüberlegungen für ausreichend:

- Für jede Abteilung ist einzeln zu prüfen, ob der Zusatzbeitrag korrekt zugeordnet wurde. Wobei die Wahl des Mitgliedsstatus frei wählbar ist, aber alle in einem Testfall vorkommen sollen. Die Testfälle TF1-TF4 in Abbildung 1 erfüllen diese Anforderung.
- Hinzu kommt der Sonderfall TF5: Passives Mitglied und keine Abteilung gewählt.
- Ein weiterer Testfall ist zu erstellen (s. Abbildung 2, TF6), bei dem alle vier Abteilungen ausgewählt werden, kombiniert mit einem beliebigen aktiven Status, z.B. Kinder/Jugendlicher. Dieser Testfall prüft, ob die Zusatzbeiträge bei Nutzung mehrerer Abteilungen korrekt berechnet werden. Da jede Abteilung einen unterschiedlichen monatlichen Beitrag (5, 11, 9, 10 Euro) erfordert, kann aus dem Ergebnis des Testfalls (Summe aller Zusatzbeiträge (35 Euro) plus Statusbeitrag (7 Euro)) abgelesen werden, ob alle Werte korrekt addiert wurden. Aus der Summe kann aber nicht erkannt werden, ob nicht zwei Beiträge vertauscht wurden (z.B. Tischtennis mit 11 statt 5 und Turnen mit 5 statt 11 Euro).

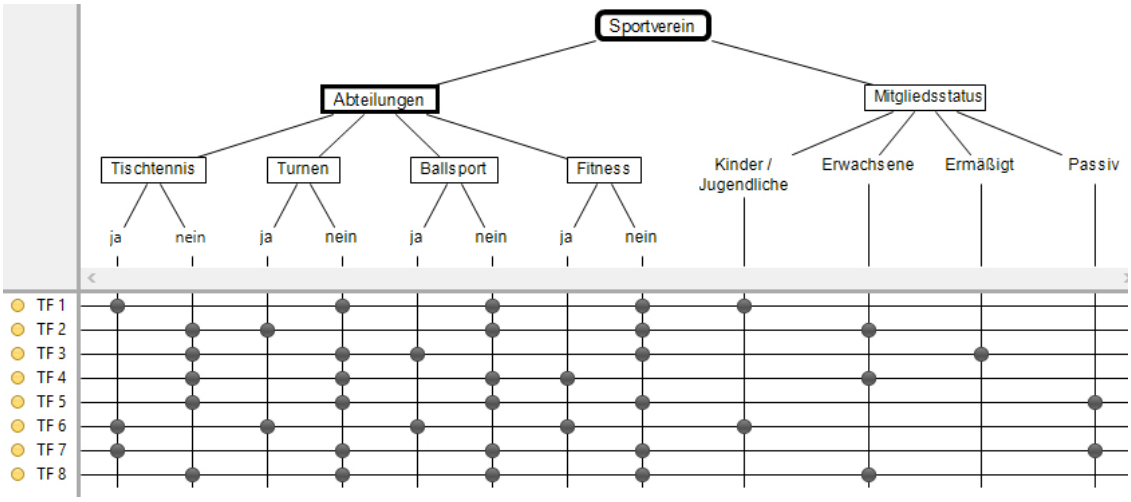

Abb. 2: Klassifikationsbaum mit acht Testfällen (TF1-TF5 identisch mit Abb.1)

Damit wäre die Hauptfunktionalität der Anwendung mit sechs Testfällen geprüft. Ergänzend können noch die zwei zusätzlichen Testfälle TF7 und TF8 durchgeführt

werden, die zu einer Fehlerbehandlung führen sollen: Ein passives Mitglied kombiniert mit einer oder mehreren Abteilungen und ein aktives Mitglied ohne Auswahl einer Abteilung. Abbildung 2 zeigt den Klassifikationsbaum, erweitert um die Testfälle TF6 - TF8.

Möglicherweise ist die Benutzungsschnittstelle so aufgebaut, dass die beiden Kombinationen für TF7 und TF8 gar nicht wählbar sind. Das kann etwa dadurch geschehen, dass als Vorbedingung der Funktion festgelegt wird, dass diese Fälle nicht auftreten dürfen. Es kann dann darauf verzichtet werden, die Eigenschaft aktiv/passiv den Mitgliedsstatus mitzugeben oder sie daraus zu ermitteln. Es ist dann Sache des Aufrufers, dafür zu sorgen (Stichwort Design by Contract).

3.1.3 8 (oder 6) Testfälle reichen aus

Mit insgesamt acht (oder sechs) Testfällen haben wir einen ausreichenden Test durchgeführt. Nun werden Sie sagen, dass aber einiges noch nicht geprüft worden ist! Ja, das stimmt. Wir stufen die Anwendung als unkritisch ein und sehen daher keine zwingende Notwendigkeit für einen umfassenden Test (alle Kombinationen der Abteilungen und der Mitgliedsstatus).

Des Weiteren gehen wir davon aus, dass eine einfache Umsetzung der Aufgabe im Programmcode erfolgte, d.h., dass mehr oder minder geradeaus programmiert wurde. Es werden also sowohl der Grundbeitrag als auch der Zusatzbeitrag separat ermittelt und es wird die jeweilige Summe gebildet. Die richtige Zuordnung der Beiträge zu den Abteilungen und zum Status wird mit den aufgeführten Testfällen geprüft.

Ein Testfall (TF6) prüft die Summenbildung bei der Zugehörigkeit zu mehreren (in dem Fall allen) Abteilungen. Wir gehen davon aus, dass auch die Summenbildung für zwei oder drei gewählte Abteilungen korrekt umgesetzt wurde. Bei einer trickreichen Programmierung, die »schräge« Fehler verursachen könnte, reichen diese wenigen Testfälle nicht aus. Wir gehen von einen guten und einfachen Programmierstil aus. Die Zeit der Programmierkünstler ist zu Recht schon lange vorbei.

Die Klassifikationsbaum-Methode wird in [1] ausführlich beschrieben, ebenso wird eine Implementierung der Mitgliedsbeiträge mit den zugehörigen Testfällen in C++ vorgestellt. Die beiden Abbildungen wurden mit dem frei verfügbaren Werkzeug Testona Light [2] für die Klassifikationsbaum-Methode erstellt.

3.2 Lean Testing mit Kombinatorischem Testen

Nehmen wir nun an, dass es doch sehr wichtig ist, wie die einzelnen Abteilungen bzw. Sportarten von den Mitgliedern gewählt - also kombiniert - werden. D.h., wir konzentrieren uns bei den folgenden Überlegungen auf Testfälle, welche die Kombination der Abteilungen (ohne Berücksichtigung des Mitgliedsstatus) prüfen. Es ergeben sich 16 mögliche Kombinationen und somit 16 Testfälle. Aber ist eine vollständige Kombination immer erforderlich? Und oft ist diese auch von der Anzahl der Testfälle gar nicht machbar, z.B. wären bei 10 unabhängigen booleschen Parametern zum Erreichen der vollständigen Kombination 1024 Testfälle durchzuführen.

Eine Lösung zur Handhabung von vielen Kombinationsmöglichkeiten ist das Kombinatorische Testen. Mit Hilfe der Mathematik kann eine sinnvolle Auswahl getroffen werden. Beim Kombinatorischen Testen wird nicht alles mit allem kombiniert, sondern es werden Parameter-Paare oder -Tripel (oder mehr), betrachtet und nur diese werden untereinander vollständig kombiniert.

Test	Tischtennis	Turnen	Ballsport	Fitness
1	nein	nein	nein	nein
2	nein	ja	ja	ja
3	ja	nein	ja	ja
4	ja	ja	nein	ja
5	ja	ja	ja	nein

Tabelle 1: Kombinatorisches Testen (2er-Tupel) mit 5 Testfällen

Sehen wir uns Tabelle 1 näher an. Auf den ersten Blick scheinen die fünf Testfälle willkürlich gewählt. Aber die fünf Testfällen umfassen alle möglichen paarweisen Kombinationen (Tischtennis/Turnen, Tischtennis/Ballsport, Tischtennis/Fitness, Turnen/Ballsport, Turnen/Fitness und Ballsport/Fitness) mit ihren jeweiligen vier Möglichkeiten (ja-ja, ja-nein, nein-ja, nein-nein).

3.2.1 Von 5 zu 9 Testfällen

Das Vorgehen wird auch als »Paarweises Testen« oder »Pairwise Testing« bezeichnet und beschränkt sich auf alle diskreten Kombinationen aller Paare (2er-Tupel) der Parameterwerte. Beim paarweisen Testen wird eine Gleichverteilung nicht garantiert, sondern nur dass jede Kombination (jedes Paar) mindestens einmal vorkommt.

Ziel des paarweisen Testens ist die Entdeckung aller Fehler, die auf der Interaktion zweier Parameter beruhen. Werden drei oder mehr möglicherweise wechselwirkende Parameter in den Kombinationen berücksichtigt, dann wird das Vertrauen in die Tests weiter verstärkt.

Tabelle 2 zeigt neun Testfälle, die das 3er-Tupel abdecken, also alle 8 Kombinationen ((ja-ja-ja, ja-ja-nein, … , nein-nein-nein) von jeweils 3 (der insgesamt 4) Parameter

Test	Tischtennis	Turnen	Ballsport	Fitness
1	nein	nein	nein	ja
2	nein	nein	ja	nein
3	nein	ja	nein	nein
4	nein	ja	ja	ja
5	ja	nein	nein	nein
6	ja	nein	ja	nein
7	ja	ja	nein	ja
8	ja	ja	ja	nein
9	ja	nein	ja	ja

Tabelle 2: Kombinatorisches Testen (3er-Tupel) mit 9 Testfällen

Die Ersparnis zu den 16 Testfällen für die vollständige Kombination scheint nicht sehr hoch zu sein, dies liegt aber an der Anzahl der zu kombinierenden Parametern, in unserem Fall vier. Bei den oben bereits erwähnten 10 booleschen Parameter reichen von den 1024 möglichen nur 13 Testfälle aus, um ein 3er-Tupel zu prüfen (nur 5 sind es beim 2er-Tupel)! Das ist eine erhebliche Reduktion der Anzahl von Testfällen.

Grundlage für den Kombinatorischen Test sind sogenannte Covering Arrays. Mit dem Verfahren kann die Anzahl der Testfälle variiert werden, je nach geforderter Intensität des Testens. Die Anzahl bleibt aber immer noch weit unter der Anzahl, wenn alle möglichen Kombinationen herangezogen werden.

Das ist Lean-Testing! Nicht alle möglichen Kombinationen, sondern eine sinnvolle, nachvollziehbare Auswahl von Testfällen wird zum Testen verwendet. Die Idee ist dabei, dass es meistens nur begrenzte Abhängigkeiten der Parameter untereinander gibt und dass man die meisten Fehler findet, wenn die Parameter wie beschrieben variiert und kombiniert werden.

Kombinatorisches Testen wird in [1] ausführlich mit Beispielen beschrieben und es wird eine Werkzeugunterstützung (Advanced Combinatorial Testing System [3]) vorgestellt, welche die entsprechenden Kombinationen erzeugt.

4. Fazit

Wir haben zwei Ansätze zum Lean Testing vorgestellt und damit verdeutlichen, dass bei systematischen Vorgehen mit wenig Aufwand ein angemessener Test durchgeführt werden kann.

In [1] finden sich viele weitere Verfahren mit ausführlichen Beispielen (programmiert in C++ [1a]), um den richtigen Mittelweg zwischen zu wenig und zu viel Testen herauszufinden. Hilfreich ist auch ein Blick in den aktuellen ISO-Standard 29119 [4], der neben den beiden vorgestellten weitere für den Unit-Test relevante Verfahren enthält.

Referenzen

[1] Andreas Spillner, Ulrich Breymann: »Lean Testing für C++-Programmierer - Angemessen statt aufwendig testen« dpunkt.verlag, 2016[2]

[1a] http://www.leantesting.de/[3] (mit freiem Download aller Programmbeispiele aus [1])

[2] http://www.testona.net/

[3] http://csrc.nist.gov/groups/SNS/acts/download_tools.html

[4] Matthias Daigl, Rolf Glunz: »ISO 29119 – Die Softwaretest-Normen verstehen und anwenden«, dpunkt.verlag, 2016

[4a] http://softwaretestingstandard.org/

[2] Teile des Beitrags sind dem Buch entnommen

[3] Die Internet-Referenzen wurden Mitte Mai 2017 geprüft