# Über die Spuren der testgetriebenen Entwicklung im Programmtext

Matthias M. Müller
Fakultät für Informatik, Universität Karlsruhe,
Am Fasanengarten 5, 76131 Karlsruhe, Germany
muellerm@ira.uka.de

**Abstract:** Der Einsatz der testgetriebenen Entwicklung verspricht einige Vorteile. So sollen zum Beispiel Programme, die mit testgetriebener Entwicklung erstellt worden sind, besser testbar sein. Bisher wurde jedoch weder beantwortet, wie sich diese bessere Testbarkeit äußern soll, noch untersucht, ob sie auch tatsächlich existiert. Um beide Fragestellungen zu untersuchen, stellen wir das Konzept der Kontrollierbarkeit von Zuweisungen vor. Wir haben Projekte, die mit testgetriebener Entwicklung erstellt wurden, und konventionelle Projekte auf ihre Eigenschaften bezüglich dieser Eigenschaft hin untersucht. So ist das Verhältnis von kontrollierbaren zu nicht kontrollierbaren Zuweisungen bei Projekten, die testgetriebene Entwicklung einsetzten, größer als bei konventionellen Projekten. Mit Hilfe der logistischen Regression untersuchten wir ebenfalls die Kontrollierbarkeit von Zuweisungen als Parameter für die Berechnung der Wahrscheinlichkeit ob testgetrieben Entwickelt wurde oder nicht. Im Vergleich zur Chidamber und Kemerer Metriksuite ist die Kontrollierbarkeit von Zuweisungen bisher der einzige signifikante Parameter in diesem Modell.

## 1 Einführung

Die testgetriebene Entwicklung [Bec02, Lin05] ist neben der Paar-Programmierung eine zentrale Technik bei Extreme Programming. Sie wurde aber im Gegensatz zur Paar-Programmierung bisher nicht so intensiv untersucht. Die zur testgetriebenen Entwicklung durchgeführten Studien [MH02, PCTV03, GW03, GSM04, EMT05] haben sich mit den Entwicklungskosten oder der Qualität der erstellten Tests auseinandergesetzt. Niemand hat bisher die Struktur, der mit testgetriebener Entwicklung hergestellten Programme, untersucht obwohl von diesen Programmen behauptet wird, sie seien “besser testbar”.

Auf der Modul-, Paket- oder Klassenebene ist das Geheimnisprinzip von Parnas [Par79] als Inbegriff gut zu testender Software Thema jedes Softwaretechniklehrplans. Testgetriebene Entwicklung arbeitet jedoch auf kleineren Ebenen, nämlich den Methoden. Können wir auch in diesen kleineren Einheiten Eigenschaften identifizieren, die eine Klasse besser testbar macht als eine andere? Ein Ansatz bietet Binder [Bin94], der Testbarkeit in Anlehnung an die Testbarkeit von Hardwareschaltungen als *kontrollierbar* und *beobachtbar* charakterisiert. Eine Methode ist kontrollierbar, wenn der Eingaberaum beherrschbar ist, und sie ist beobachtbar, wenn ihr Ergebnis dem Test zugänglich gemacht werden kann.

Bei einer nicht kontrollierbaren Methode ist unklar, welche Ursache eine bestimmte Ausgabe hatte. Bei einer nicht beobachtbaren Methode, können die Folgen einer bestimmten Eingabe nicht nachvollzogen werden.

Dieses Papier überträgt das Konzept der Kontrollierbarkeit auf Zuweisungen. Kontrollierbarkeit auf Zuweisungsebene bedeutet hierbei, dass die Operanden der rechten Seite der Zuweisung durch die Eingabeparameter der Methode festgelegt oder transitiv durch diese berechnet werden können. Diese *Zuweisungs-Kontrollierbarkeit* wird nachfolgend daraufhin untersucht, ob sie als Indikator für den Gebrauch der testgetriebenen Entwicklung benutzt werden kann. Wir gehen davon aus, das die testgetriebene Entwicklung zu Methoden führt, die mehr kontrollierbare Zuweisungen besitzen als Methoden, die konventionell erstellt wurden. Dazu vergleichen wir die Struktur von fünf Projekten, die mit testgetriebener Entwicklung erstellt wurden, mit der Struktur von drei Opensource-Projekten. Als Vergleichsmetriken verwenden wir die von Chidamber und Kemerer [CK94] vorgeschlagene Metriksuite für objektorientierte Programme.

## 2 Metrik zur Testbarkeit

### 2.1 Kontrollierbarkeit

Wikipedia beschreibt das Konzept der Kontrollierbarkeit wie folgt:

> Das Konzept der Kontrollierbarkeit beschreibt die Eigenschaft eines Systems, durch die das System nur durch zulässige Eingaben in jeden gültigen Zustand seines Zustandsraumes gebracht werden kann [Übers. d. Autors].

Übertragen auf Objekte in der objektorientierten Welt bedeutet Kontrollierbarkeit, dass alle Eingabeparameter bekannt sind und dass sie auch so manipuliert werden können, dass das Verhalten des Objektes lediglich anhand der Eingaben beschrieben werden kann. Da sich der Zustand eines Objektes durch Zuweisungen an lokale, Instanz- oder Klassenvariablen verändert, konzentrieren wir uns auf die Analyse der Zuweisungen. Aufrufe von Methoden, die keinen Rückgabewert liefern werden bei der Berechnung der Kontrollierbarkeit momentan ignoriert.

### 2.2 Kontrollierbarkeit von Zuweisungen

Die Berechnung der Kontrollierbarkeit von Zuweisungen ist ein vorwärtsgerichtetes Datenflussproblem. Zunächst einmal sind alle Methodenparameter sowie private oder öffentliche Instanz- und Klassenvariablen kontrollierbar. Mit Hilfe dieser Elemente wird schließlich anhand Tabelle 1 die Kontrollierbarkeit einer Zuweisung berechnet. Das Ergebnis einer Zuweisung ist kontrollierbar, wenn die rechte Seite kontrollierbar ist. Damit ein Ausdruck kontrollierbar ist, müssen alle Bezeichner, die in diesem Ausdruck vorkommen,

Tabelle 1: Kontrollierbarkeit von Operationen

| **Operation** | **Kontrollierbarkeit des Ergebnisses** |
|---|---|
| $lhs := rhs$ | Linke Seite einer Zuweisung kontrollierbar, wenn rechte Seite kontrollierbar. |
| $exp_1 \oplus exp_2$ | Ergebnis beliebiger binärer Operation $\oplus$ kontrollierbar, wenn beide Operanden $exp_1$ und $exp_2$ kontrollierbar. |
| $\oplus\ exp_1$ | Ergebnis beliebiger unärer Operation $\oplus$ kontrollierbar, wenn Operand $exp_1$ kontrollierbar. |
| $obj.foo\,(\,a,\,b\,)$ | Ergebnis eines Funktionsaufrufs kontrollierbar, wenn $obj$ und Parameter $a$ und $b$ kontrollierbar. |

ebenfalls kontrollierbar sein. Ein Spezialfall bildet eine bedingte Zuweisung, siehe Abbildung 1. So ist der Bezeichner $a$ in Zeile 6 nur kontrollierbar, wenn entweder *beide*

```
1  if ( cond ) {
2    a = exp1;
3  } else {
4    a = exp2;
5  }
6  b = ... a ...
```

Abbildung 1: Bedingte Zuweisung

Ausdrücke $exp_1$ und $exp_2$ in den Zeilen 2 und 4 kontrollierbar sind oder die Bedingung $cond$ aus Zeile 1 kontrollierbar ist und einer der Ausdrücke $exp_1$ oder $exp_2$.

*Nicht* kontrollierbare Elemente sind alle möglichen Konstanten sowie alle Aufrufe an `this`.

### 2.3 Berechnung

Die Kontrollierbarkeit einer Methode $m$ ergibt sich aus dem Verhältnis von kontrollierbaren zu allen Zuweisungen einer Methode. Wir nennen diese Metrik *Zuweisungskontrollierbarkeit* (*engl. statement controllability, StatCon*).

$$\mathsf{StatCon}(m) = \frac{\mathsf{Anzahl\ kontrollierbarer\ Zuweisungen\ in\ Methode}\ m}{\mathsf{Anzahl\ aller\ Zuweisungen\ in\ Methode}\ m}$$

Ihr Wertebereich liegt zwischen 0 und 1. Um die Kontrollierbarkeit einer Klasse $c$ zu bestimmen, verwenden wir das arithmetische Mittel der Zuweisungskontrollierbarkeit für

Methoden. Für eine Klasse $c$ mit $n$ Methoden $m_i$ ($i = 1 \ldots n$) ist

$$\mathsf{StatCon}(c) = \frac{1}{n} \sum_{i=1}^{i=n} \mathsf{StatCon}(m_i) \tag{1}$$

Methoden ohne eine Zuweisung werden bei der Mittelwertberechnung nicht mitbetrachtet.

Implementiert wurde StatCon auf Bytecodeebene mit Hilfe der *Byte Code Engineering Library* (BCEL) [BCE] des Jakarta Apache Projekts.

## 3 Die Vergleichsmetriken

Die in 2 vorgestellte Metrik StatCon wird mit den folgenden acht Metriken verglichen. Von diesen Metriken bilden die letzten sechs die von Chidamber und Kemerer [CK94] vorgeschlagene Metriksuite für objektorientierte Programme.

**Assign** Anzahl aller Zuweisungen einer Klasse. Auf Methodenebene werden die Anzahl der Zuweisungen pro Methode gezählt.

**Size** Anzahl aller Byte-Code-Anweisungen einer Klasse. Auf Methodenebene werden die Anzahl der Anweisungen pro Methode gezählt.

**WMC** Anzahl der gewichteten Methoden pro Klasse: $WMC = \sum_{i=1}^{|Methods|} c_i$. Da die Gewichte $c_i$ auf eins gesetzt werden, entspricht dies der Anzahl der öffentlichen Methoden der Klasse.

**DIT** Tiefe der Klasse im Vererbungsbaum. Die Wurzel des Baumes hat die Tiefe eins.

**NOC** Anzahl der Kinder einer Klasse. Sie ist die Zahl der direkten Unterklassen einer Klasse.

**CBO** Die Kopplung einer Klasse $K$ zu anderen Klassen ist die Anzahl anderer Klassen, deren Methoden oder Instanzvariablen von $K$ benutzt werden.

**RFC** Summe der verschiedenen Methoden, die innerhalb einer Klasse direkt aufgerufen werden (*engl. response for a class*). Es werden direkte Aufrufe eigener und fremder Methoden gezählt.

**LCOM** Mangel an Zusammenhang zwischen Methoden: $LCOM = \max(|N| - |S|, 0)$. Dabei steht $N$ für die Menge von Methodenpaaren *ohne* eine gemeinsam benutzte Instanzvariable (*engl. no sharing*) und $S$ für die Menge von Methodenpaaren *mit* mindestens einer gemeinsam benutzten Instanzvariable (*engl. sharing*).

# 4 Die Projekte

Dieser Abschnitt gibt zunächst einen Überblick über die verwendeten Projekte. Eine Diskussion über mögliche Nebeneffekte verursacht durch diese Projekte schließt sich an.

## 4.1 Übersicht

Wir haben fünf Projekte, die nachweislich mit testgetriebener Entwicklung erstellt wurden, mit drei konventionellen Projekten verglichen. Die Tabelle 2 gibt einen Überblick über die verwendeten Projekte, ob sie mit testgetriebener Entwicklung erstellt wurden und ihre Größe in Anzahl Klassen und Pakete. Webtest [Web] ist ein Testwerkzeug für

Tabelle 2: Projekte in der Übersicht

| | | Anzahl | |
|---|---|---|---|
| **Name** | **TGE** | **Klassen** | **Paketen** |
| Webtest | ja | 149 | 21 |
| XPChess1 | ja | 63 | 8 |
| XPChess2 | ja | 48 | 8 |
| XPChess3 | ja | 68 | 8 |
| Yaps | ja | 100 | 16 |
| Summe | | 428 | 61 |
| Ant | nein | 372 | 22 |
| JUnit | nein | 75 | 7 |
| Log4j | nein | 228 | 19 |
| Summe | | 675 | 48 |

Webapplikationen. XPChess1 bis XPChess3 sind die Praktikumsprogramme der Gruppen des Extreme Programming Praktikums im Sommersemester 2005. Diese Programme sind Schachprogramme mit textueller Ein- und Ausgabe. Yaps ist ein Portalrahmenwerk eines mittelständischen Unternehmens, mit dem kleine Webapplikationen zu einem Portal zusammengebaut werden können. Ant [Ant] ist die Apache-Erweiterung des UNIX-Werkzeuges Make. JUnit [JUn] ist die für testgetriebene Entwicklung erforderliche Java-Variante der xUnit-Familie. Log4j [Log] ist die Java-Variante des Protokollrahmenwerkes von Apache. Die Größenangaben aller Projekte beziehen sich auf die reinen Applikationsquellen, d. h. die Quellen der Unit-Tests wurden nicht gezählt und später auch nicht weiter ausgewertet.

### 4.2 Gültigkeit

Es gibt mehrere Gefahren bezüglich der Gültigkeit der Ergebnisse. Zunächst einmal ist der Datensatz der testgetriebenen Projekte kleiner als der Datensatz der konventionellen Projekte. Dies liegt vor allem daran, dass wir bisher Probleme hatten industrielle Projekte zu bekommen, die mit testgetriebener Entwicklung erstellt wurden. Dieser Mangel erklärt auch, weshalb wir die drei Studentenprojekte XPChess1 bis XPChess3 mit in die Auswertung einbezogen haben. Damit ist zwar der Datensatz der testgetriebenen Projekte etwas größer geworden, doch haben wir nun drei Projekte, die zwar von unterschiedlichen studentischen Gruppen entwickelt wurden aber aus dem gleichen Problemumfeld stammen.

Ein weiteres Problem ergibt sich aus diesem Behelf. Wir studieren Programme, die von Studenten und nicht von Entwicklern geschrieben wurden, die Erfahren in der Anwendung der testgetriebenen Entwicklung sind. Die Studenten haben in der Anfangszeit der testgetriebenen Entwicklung zunächst Probleme [Wil01, MLSM04], die sich aber mit zunehmender Dauer der Lehrveranstaltung wieder legen. Es ist bisher unbekannt, inwieweit sich die Programme der Entwickler, die in testgetriebenen Entwicklung unerfahren sind, von den Programmen der Entwickler unterscheiden, die in testgetriebener Entwicklung erfahren sind. Daher kann es sein, dass die gezeigten Unterschiede in diesen Programmen nicht allein auf die testgetriebenen Entwicklung sondern auch auf den unterschiedlichen Kenntnisstand der Programmautoren zurückgeführt werden können.

## 5 Resultate

### 5.1 Die Projekte aus der Sicht der Metriken

Bevor wir uns den Eigenschaften der testgetriebenen Entwicklung zuwenden, betrachten wir zunächst die Projekte aus der Sicht der 9 Metriken. Tabelle 3 gibt einen Überblick über die Metrikwerte der beiden Projektklassen: konventionell (kon) und testgetrieben (TGE). Die Tabelle zeigt für jede Metrik und jede Gruppe das Minimum (0 % Quantil), den Median (50 % Quantil), das Maximum (100 % Quantil) und den Mittelwert ($\overline{x}$). Mit dem Rangsummentest von Wilcoxon [BT94, 131 ff.] haben wir die Stichproben beider Gruppen für je eine Metrik auf Unterschiede hin untersucht. Die letzte Spalte der Tabelle 3 zeigt die p-Werte. Werte unterhalb des 5-Prozent Niveaus sind fett gedruckt.

Es fällt auf, dass bis auf die Tiefe im Vererbungsbaum (DIT) und die Anzahl der Methoden in einer Klasse (WMC), sich die beiden Projektgruppen in allen anderen Metriken unterscheiden. Die gemessenen Unterschiede könnten mit dem Einsatz der testgetriebenen Entwicklung zusammenhängen. Abschnitt 4.2 diskutiert jedoch noch weitere Faktoren, die einen Einfluss auf die Messungen haben könnten, sodass es vermessen wäre den Unterschied alleinig auf die testgetriebene Entwicklung zurückzuführen.

Tabelle 3: Die Projekte aus der Sicht der Metriken.

| | konventionell | | | | TGE | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Quantile in % | | | | Quantile in % | | | | Wilcoxon |
| **Metrik** | **0** | **50** | **100** | $\bar{x}$ | **0** | **50** | **100** | $\bar{x}$ | **p-Wert** |
| StatCon | 0 | 0.42 | 1 | 0.45 | 0 | 0.51 | 1 | 0.54 | $<$**0.01** |
| LCOM | 0 | 0 | 741 | 3.37 | 0 | 1 | 325 | 7.28 | $<$**0.01** |
| RFC | 1 | 10 | 197 | 14.35 | 1 | 8 | 91 | 11.26 | $<$**0.01** |
| CBO | 2 | 8 | 165 | 10.81 | 2 | 6 | 60 | 8.5 | $<$**0.01** |
| DIT | 1 | 2 | 11 | 2.15 | 1 | 1.5 | 10 | 2.16 | 0.32 |
| NOC | 0 | 0 | 52 | 0.35 | 0 | 0 | 31 | 0.48 | $<$**0.01** |
| WMC | 1 | 5 | 133 | 8.29 | 1 | 4 | 59 | 6.95 | 0.6 |
| Assign | 0 | 5 | 273 | 14.11 | 0 | 3 | 239 | 6.95 | $<$**0.01** |
| Size | 2 | 67 | 2178 | 140.18 | 3 | 57 | 1762 | 90.65 | $<$**0.01** |

### 5.2 StatCon auf Methodenebene

Nun wollen wir uns die StatCon-Werte auf Methodenebene für jedes Projekt ansehen. Abbildung 2 zeigt die Histogramme für jedes Projekt. Zwei Eigenschaften fallen auf. Zunächst einmal häufen sich die StatCon-Werte bei 0 und 1. Jedes Projekt hat sehr viele Methoden, die entweder gar keine kontrollierbare Zuweisung enthalten (StatCon=0,linker großer Balken in jedem Histogramm) oder nur kontrollierbare Zuweisungen enthalten (StatCon=1, rechter großer Balken in jedem Histogramm. Ein zweites Merkmal offenbart sich, wenn die Höhe der beiden Balken verglichen wird. So hat jedes konventionelle Projekt mehr Methoden mit nur *un*kontrollierbaren Zuweisungen als Methoden, in denen *jede* Zuweisung kontrollierbar ist. Die linken Balken sind für die konventionellen Projekte größer als die rechten Balken. Diese Beobachtung gilt bei den testgetriebenen Projekten nur für Webtest. Tabelle 4 zeigt hierzu den Anteil der Methoden, die mindestens eine nicht-kontrollierbare Zuweisung besitzen, im Vergleich zu den Methoden, die nur kontrollierbare Zuweisungen enthalten. Der Anteil an Methoden, die nur kontrollierte Zuweisungen enthalten, liegt bei den testgetriebenen Projekten bei 43,5 Prozent, siehe zweite Spalte von rechts in der Zeile markiert mit TGE. Bei den konventionellen Projekten sind dies nur 33,6 Prozent der Methoden. Bei den testgetriebenen Projekten liegt das Verhältnis von Methoden die nur kontrollierte Zuweisungen haben im Vergleich zu den verbleibenden Methoden bei 484/628 = 0.771. Für die konventionellen Projekte fällt dieses Verhältnis mit 964/1909 = 0.505 geringer aus. Für die gesamte Stichprobe liegt dieses Verhältnis bei 1448/2537 = 0.571. Das Verhältnis ist für testgetriebene Projekte 0.771/0.505 = 1.526 mal größer als für konventionelle Projekte.

Tabelle 4: Anteil an kontrollierbaren Methoden pro Projekt und Projektgruppe.

| | Methoden mit | | | | |
|---|---|---|---|---|---|
| | StatCon $< 1$ | | StatCon $= 1$ | | |
| **Name** | **Anzahl** | **%** | **Anzahl** | **%** | **Insgesamt** |
| Webtest | 353 | 58.3 | 253 | 41.7 | 606 |
| XPChess1 | 46 | 53.5 | 40 | 46.5 | 86 |
| XPChess2 | 39 | 50.6 | 38 | 49.4 | 77 |
| XPChess3 | 52 | 47.7 | 57 | 52.3 | 109 |
| Yaps | 138 | 59.0 | 96 | 41.0 | 234 |
| **TGE** | 628 | 56.5 | 484 | 43.5 | 1112 |
| Ant | 1144 | 63.1 | 669 | 36.9 | 1813 |
| JUnit | 146 | 68.9 | 66 | 31.1 | 212 |
| Log4j | 619 | 73.0 | 229 | 27.0 | 848 |
| **konv** | 1909 | 66.4 | 964 | 33.6 | 2873 |
| **Alle** | 2537 | 63.7 | 1448 | 36.3 | 3985 |

### 5.3 Korrelationsanalyse auf Klassenebene

Dieser Abschnitt beschäftigt sich mit der Frage, ob und wie stark StatCon mit den anderen 9 Metriken korreliert ist. Für führten dazu eine Korrelationanalyse nach Spearman [BT94, 232 ff.] durch. Tabelle 5 zeigt die Korrelationskoeffizienten für die jeweiligen Datensätze. Die Spalte mit der Überschrift Alle zeigt die Ergebnisse, wenn alle Projekte gemeinsam

Tabelle 5: Korrelationen auf Klassenebene

| | StatCon | | |
|---|---|---|---|
| | **Alle** | **TGE** | **konv** |
| Assign | -0.30 | -0.19 | -0.32 |
| Size | -0.34 | -0.32 | -0.35 |
| WMC | -0.20 | -0.19 | -0.26 |
| DIT | -0.22 | -0.26 | -0.20 |
| NOC | -0.07 | -0.22 | 0.02 |
| CBO | -0.27 | -0.20 | -0.30 |
| RFC | -0.32 | -0.31 | -0.32 |
| LCOM | -0.23 | -0.21 | -0.29 |

ausgewertet werden. Die beiden folgenden Spalten beschränken sich auf die testgetriebenen beziehungsweise nur auf die konventionellen Projekte. Zwei Eigenschaften fallen auf. Zum einen sind die ermittelten Werte alle kleiner oder gleich 0,35. Das lässt auf eine sehr geringe Korrelation schließen. Zum anderen ist StatCon in den Datensätzen Alle und TGE zu jeder anderen Metrik negativ korreliert.

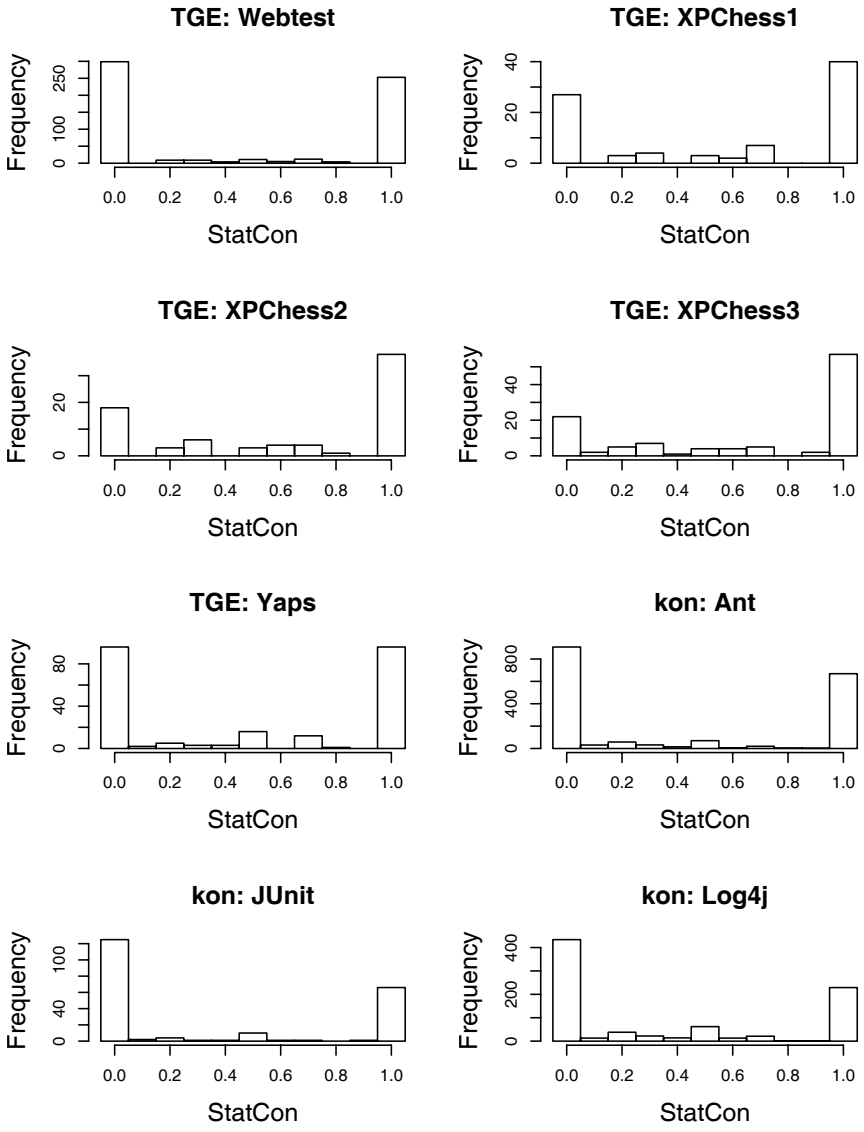


Abbildung 2: Verteilung von StatCon auf Methodenebene über die Projekte.

Die negative Korrelation zur Klassengröße (Size) bedeutet zum Beispiel, dass kleinere Klassen eine Tendenz haben mehr kontrollierbare Zuweisungen zu besitzen als größere Klassen. Wenn wir den Grad der Kontrollierbarkeit als Indikator für die Testbarkeit benutzen, so sind kleinere Klassen tendenziell besser testbar als größere Klassen. Ähnliches gilt für Klassen mit geringerer Tiefe im Vererbungsbaum (siehe DIT) oder auch für Klassen mit geringerer Kopplung (CBO). Hier scheint StatCon die Faustregeln für gute Testbarkeit zu unterstützen.

### 5.4 Logistische Regression

In diesem letzten Abschnitt untersuchen wir die Eigenschaft von StatCon als Prediktor für den Einsatz von testgetriebener Entwicklung. Dazu verwenden wir die logistische Regression, siehe z. B. Kleinbaum [Kle94]. Die logistische Regression ist eine Verallgemeinerung der linearen Regression auf Datensätze mit einem nominalen Wertebereich. Die Eigenschaft eines Projektes mit testgetriebener Entwicklung erstellt worden zu sein, wird mit Hilfe der nominalen Variable TDD modelliert. Klassen, die mit testgetriebener Entwicklung erstellt wurden, werden mit TDD=1 alle anderen Klassen mit TDD=0 kodiert. Das logistische Modell sieht wie folgt aus:

$$P(TDD = 1|X_1, \ldots, X_9) = \frac{1}{1 + e^{-f(X_1,\ldots,X_9)}}$$

$$f(X_1, \ldots, X_9) = \alpha + \sum_{i=1}^{9} \beta_i X_i$$

Für eine bessere Übersicht haben wir die Metriknamen durch die Variablennamen $X_i$ ($i = 1, \ldots, 9$) ersetzt. Wir suchen also Parameterwerte, mit denen die Wahrscheinlichkeit berechnet werden kann, ob eine Klasse mit testgetriebener Entwicklung erstellt wurde oder nicht. Hier interessieren uns die konkreten Werte für $\alpha$ und die $\beta_i$ jedoch nicht. Nein, wir sind daran interessiert, ob und welche der beteiligten Metriken signifikant zum Modell beiträgt. Wir schätzen die Parameter ($\beta_i$ und $\alpha$) für zwei Datensätze. Im Datensatz $D_{Alle}$ sind alle Klassen enthalten. Im Datensatz $D_{Assign>0}$ sind alle Klassen enthalten, die mindestens eine Zuweisung in einer Methode enthalten.

Tabelle 6 gibt für jeden Datensatz einen Überblick über die geschätzten Parameter, den Standardfehler und den p-Wert für den Test der Null-Hypothese, dass der entsprechende Parameter *nicht* entscheidend am Modell beteiligt ist. Interessant für diese Analyse sind die p-Werte bei jedem Parameter. Es fällt auf, dass lediglich $\alpha$ und StatCon bei beiden Parametern mit einer Signifikanz von weniger als einem Promill zum Modell beitragen. Die Anzahl der Zuweisungen pro Klasse (Assign) ist noch im Datensatz $D_{Alle}$ signifikant. Ansonsten liegt der p-Wert bei allen anderen Parametern deutlich über dem 10 Prozent Niveau.

Betrachten wir lediglich die Klassen die mindestens eine Zuweisung enthalten, so können wir als Fazit festhalten, dass StatCon die Spuren testgetriebener Entwicklung im Programmtext besser als alle bisher betrachteten Metriken zu beschreiben scheint.

Tabelle 6: Parameterschätzungen für das logistische Modell.

| | $\mathbf{D}_{\mathbf{Alle}}$ | | | $\mathbf{D}_{\mathbf{Assign>0}}$ | | |
|---|---|---|---|---|---|---|
| **Parameter** | **Geschätzt** | **Std. Fehl.** | **p-Wert** | **Geschätzt** | **Std. Fehl.** | **p-Wert** |
| $\alpha$ | -1.1955 | 0.2042 | $<$**0.001** | -0.7337 | 0.2129 | $<$**0.001** |
| StatCon | 0.7171 | 0.2136 | $<$**0.001** | 0.8884 | 0.2306 | $<$**0.001** |
| Assign | -0.0393 | 0.0123 | **0.001** | 1.9888 | 62.9964 | 0.974 |
| Size | 0.0013 | 0.0013 | 0.312 | -1.9909 | 62.9964 | 0.974 |
| WMC | 0.0134 | 0.0160 | 0.402 | 0.0159 | 0.0161 | 0.324 |
| DIT | 0.0609 | 0.0503 | 0.225 | -0.0034 | 0.0533 | 0.949 |
| NOC | 0.0245 | 0.0294 | 0.404 | 0.0246 | 0.0308 | 0.424 |
| CBO | -0.0006 | 0.0209 | 0.975 | 0.0163 | 0.0226 | 0.468 |
| RFC | 0.0082 | 0.0201 | 0.683 | -0.0058 | 0.0203 | 0.773 |
| LCOM | 0.0070 | 0.0051 | 0.169 | 0.0069 | 0.0051 | 0.180 |

## 6 Zusammenfassung und Ausblick

Wir haben in dieser Studie die Kontrollierbarkeit von Zuweisungen StatCon im Programmtext untersucht. Wir verglichen die Programmtexte konventioneller Projekte mit Projekten, die mit testgetriebener Entwicklung erstellt wurden. Die Analyse zeigte folgende Ergebnisse:

- Der Anteil der Methoden mit StatCon gleich 1 ist bei testgetriebenen Projekten größer als bei konventionellen Projekten.
- StatCon ist zu allen anderen untersuchten Metriken negativ korreliert. Diese Eigenschaft spiegelt das intuitive Verständnis von Testbarkeit wieder.
- Bei der Bestimmung des Modells zur Berechnung der Wahrscheinlichkeit, mit der ein Projekt mit testgetriebener Entwicklung erstellt wurde, stellt StatCon bei Klassen mit mindestens einer Zuweisung bisher den einzigen signifikanten Parameter dar.

Diese Studie stellt erst den Anfang einer eingehenden Untersuchung der Spuren der testgetriebenen Entwicklung im Programmtext dar, denn für endgültige Ergebnisse ist unser Datensatz noch zu klein und nicht repräsentativ genug.

## 7 Danksagung

Ich möchte mich an dieser Stelle bei Johannes Link und Matthias Grund für Ihren Einsatz bei der Beschaffung von Projekten, die mit testgetriebener Entwicklung erstellt wurden, und bei Christian Frommeyer für die Implementierung von StatCon bedanken.

## Literatur


[Ant] Ant. http://ant.apache.org/.

[BCE] Byte Code Engineering Library (BCEL). http://jakarta.apache.org/bcel/index.html.

[Bec02] K. Beck. *Test Driven Development: By Example*. Addison-Wesley, 2002.

[Bin94] R. Binder. Design for Testability in Object-Oriented Systems. *Communications of the ACM*, 37(9):87–101, September 1994.

[BT94] H. Büning und G. Trenkler. *Nichtparametrische statistische Methoden*. de Gruyter, 2.. Auflage, 1994.

[CK94] S. Chidamber und C. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, Juni 1994.

[EMT05] H. Erdogmus, M. Morisio und M. Torchiano. On the Effectiveness of the Test-First Approach to Programming. *IEEE Transactions on Software Engineering*, 31(3):226–237, Marz 2005.

[GSM04] A. Geras, M. Smith und J. Miller. A Prototype Empirical Evaluation of Test Driven Development. In *International Symposium on Software Metrics (Metrics)*, Seiten 405–416, Chicago, Illinois, USA, September 2004.

[GW03] B. George und L. Williams. An initial investigation of test driven development in industry. In *ACM symposium on Applied computing*, Seiten 1135–1139, Melbourne, Florida, USA, 2003.

[JUn] JUnit. http://www.junit.org/.

[Kle94] D. Kleinbaum. *Logistic regression: a self-learning text*. Springer, 94.

[Lin05] J. Link. *Softwaretests mit JUnit*. dpunkt.verlag, 2. Auflage, 2005.

[Log] Log4j. http://logging.apache.org/.

[MH02] M. Müller und O. Hagner. Experiment about test-first programming. *IEE Proceedings Software*, 149(5):131–136, Oktober 2002.

[MLSM04] M. Müller, J. Link, R. Sand und G. Malpohl. Extreme Programming in Curriculum: Experiences from Academia and Industry. In *Conference on Extreme Programming and Agile Processes in Software Engineering (XP2004)*, Seiten 294–302, Garmisch-Partenkirchen, Germany, Juni 2004.

[Par79] D. Parnas. Designing software for ease of extension and contraction. *IEEE Transactions on Software Engineering*, 5(2):128–138, Marz 1979.

[PCTV03] M. Pancur, M. Ciglaric, M. Trampus und T. Vidmar. Towards empirical evaluation of test-driven development in a university environment. In *EUROCON 2003. Computer as a Tool. The IEEE Region 8.*, Jgg. 2, Seiten 83–86, September 2003.

[Web] Webtest. http://webtest.canoo.com.

[Wil01] D. Wilson. Teaching XP: A Case Study. In *XP Universe*, Raleigh, NC, USA, Juli 2001.