# Automatische Erkennung von Model Smells in Simulink-Modellen



# Automatische Erkennung von Model Smells in Simulink-Modellen

Jonas Paul Winkler, Quang Minh Tran

jwinkler@mailbox.tu-berlin.de, tu-berlin.tran@daimler.com

**Abstract:** Simulink-Modelle werden mit zunehmender Komplexität anfällig für Qualitätsdefizite. Die Ursache dafür sind strukturelle Probleme, sogenannte *Model Smells*. Die manuelle Erkennung von Model Smells ist aufwändig, daher wird ein Verfahren zur automatisierten Erkennung von Model Smells vorgestellt. Dieses ermöglicht es Modellierern, effizient die Qualität von Modellen zu verbessern.

## 1 Einführung

Matlab/Simulink von The MathWorks ist ein Werkzeug zur modellgetriebenen Softwareentwicklung. Simulink wird insbesondere in der Automobilbranche zur Entwicklung der Software für Microcontroller eingesetzt.

Bedingt durch die stetig steigende Komplexität der umzusetzenden Funktionen und sich während der Entwicklung ändernden Anforderungen sinkt in großen Simulink-Modellen über längere Zeit die interne Qualität [KKT11, S. 194]. Dies führt zu schlechter Wartbarkeit, Erweiterbarkeit und Testbarkeit des Modells.

Die Ursachen von Qualitätsdefiziten werden in der Regel manuell gesucht und behoben. Dieser Prozess ist fehleranfällig und sehr zeitaufwändig. Daher ist es wünschenswert, die Analyse der internen Modellqualität und die Beseitigung von Defiziten durch Werkzeuge zu automatisieren, wie es bereits bei imperativen Programmiersprachen der Fall ist. Dort gibt es umfangreiche Analysewerkzeuge[1] und in die Entwicklungsumgebungen integrierte Refactoring-Tools zum Verbessern von schlechtem Quellcode.

Ziel der Arbeit ist die Schaffung einer Grundlage zur Erkennung von strukturellen, qualitätsmindernden Problemen in Simulink-Modellen und die Entwicklung eines Prototyps zur Demonstration der Umsetzbarkeit der gezeigten Grundlagen.

## 2 Verwandte Arbeiten

Strukturelle und qualitätsmindernde Probleme existieren auch in imperativen Programmiersprachen. Dort sind sogenannte Code Smells ein Indiz für geringe interne Codequa-

[1] SonarQube, http://www.sonarqube.org/

lität [Fow99, S. 67ff]. Diese dienen in dieser Arbeit als Ausgangspunkt. Beispiele für bekannte Smells sind „Lange Methode“, „Große Klasse“ und „Duplizierter Code“. Code Smells können durch gezielten Einsatz von Refactoring behoben werden.

In Simulink gibt es bereits unterschiedlich Ansätze zur Qualitätsanalyse von Modellen. Jan Scheible beschreibt ein auf Metriken basierendes Qualitätsmodell für Simulink-Modelle [Sch12]. In diesem werden Kennzahlen (Anzahl der Blöcke, Durchschnittliche Anzahl an Subsystemen pro Subsystem) zu einer quantitativen Qualitätsbewertung von Simulink-Modellen zusammengesetzt. In dieser Arbeit überführen wir die Idee der Code Smells nach Simulink und führen den neuen Begriff „Model Smell“ ein. Zur Beschreibung von Model Smells nutzen wir unter anderem die Metriken aus [Sch12].

## 3 Model Smells

Ein Model Smell ist ein allgemeines Muster, dessen Vorhandensein in einem Simulink-Modell ein Indiz für geringe interne Modellqualität ist. Ein Simulink-Modell besteht aus Blöcken und gerichteten Linien. Blöcke repräsentieren die Operationen in einem Modell, Linien verbinden mehrere Blöcke miteinander. Über Linien werden Signale zwischen Blöcken transportiert.

Model Smells und Code Smells haben folgende Eigenschaften gemeinsam:

- Die Existenz eines Smells muss nicht zwingend niedrige Qualität bedeuten, daher muss der Entwickler für jeden in einem Artefakt (Code, Modell) gefundenen Smell selbst entscheiden, ob der Smell tatsächlich ein Problem darstellt.
- Wenn ein Problem erkannt wurde, muss dieses durch geeignete Gegenmaßnahmen („Refactoring“) eliminiert werden. Dabei verändert sich die Funktionalität des Artefakts nicht.

Das Muster eines Model Smells besteht aus auf abstrakten Simulink-Elementen formulierten *Bedingungen* und zwischen diesen Simulink-Elementen bestehenden *Zusammenhängen*. Mögliche Simulink-Elemente sind insbesondere Blöcke, Linien und Signale. Treffen alle Bedingungen und Zusammenhänge auf einen konkreten Teil eines Modells zu, so stellt dieser Teil im gegebenen Modell eine *Instanz* des Model Smells dar.

Nachfolgend werden beispielhaft zwei Model Smells vorgestellt.

**Großes Subsystem.** Dieser auf Basis des Code Smells „Lange Methode“ [Fow99, S. 69] entwickelte Model Smell erfasst Subsysteme, die mehr als eine festgelegte Anzahl an Kindblöcken besitzen. Sehr große Subsysteme implementieren meist zu viele Funktionen und sind daher ein Hinweis auf schlechte Modularisierung des Systems.

**Signalumbenennung.** Werden Signale entlang ihres Pfades umbenannt, wird der Signalname überschrieben. Für den Modellierer wird es dadurch schwieriger, Ursprung und Bedeutung von umbenannten Signalen zu bestimmen. Instanzen dieses Smells verschlechtern die Lesbarkeit und Verständlichkeit eines Modells.[2]

[2] http://www.mathworks.de/de/help/simulink/mdl_gd/signals-and-signal-labels.html

# 4 Formulierung in Prolog

Um Model Smells automatisiert finden zu können werden Simulink-Modelle und Model Smells in Prolog dargestellt. Modelle werden durch Fakten und Model Smells durch Regeln beschrieben. Prolog ermöglicht es daraufhin, im Modell automatisiert nach Model Smells zu suchen.

Folgende Konventionen gelten bei der Darstellung von Simulink-Modellen durch Fakten:

| | |
|---|---|
| `block(b).` | Legt fest, dass `b` ein Simulink-Block ist. |
| `block_type(b,t).` | Der Typ des Blocks `b` ist `t`. `t` ist ein String. |
| `block_parent(b,p).` | `p` ist ein Subsystem und enthält den Block `b`. |
| `line(l).` | `l` ist eine Simulink-Linie. |
| `line_source(l,b).` | Die Linie `l` startet an Block `p`. |
| `line_name(l,n).` | Die Linie `l` trägt den Namen `n` (String). |
| `signal_source(s,b).` | Das Signal `s` wird an Block `p` erzeugt. |

Fakten für ein konkretes Simulink-Modell werden mithilfe eines Skripts automatisch generiert. Auf Basis dieser Konventionen können nun die in Abschnitt 3 eingeführten Model Smells durch Regeln beschrieben werden.

```
large_subsystem(X) :-
    block(X),
    block_type(X, 'SubSystem'),
    aggregate_all(count, block_parent(_, X), C),
    C >= 15.
```

Wenn `X` ein Simulink-Block, insbesondere ein SubSystem-Block ist und die Anzahl der Blöcke, die `X` als Vater haben, größer oder gleich 15 ist, so ist das durch `X` repräsentierte Subsystem eine Instanz des Smells *Großes Subsystem*.

```
signal_rename(S, L) :-
    signal_source(S, Block),
    line_source(StartLine, Block),
    line_signal(L, S),
    line_name(L, _),
    \+ StartLine = L.
```

Die Quelle des Signals `S` ist der Block `Block`. Mit diesem Block verbunden ist die Linie `StartLine` - die erste Linie, auf der `S` geführt wird. Existiert eine Linie `L`, die ebenfalls das Signal `S` führt (ausgedrückt durch `line_signal`), einen beliebigen Namen trägt und sich von `StartLine` unterscheidet, so wird das Signal `S` durch die Linie `L` umbenannt. Die Kombination von `S` und `L` ist eine Instanz des Smells *Signalumbenennung*.

Gegeben sei ein durch Fakten repräsentiertes Simulink-Modell. Dann können mithilfe der als Regeln ausgedrückten Model Smells und einer Prolog-Engine zwei Operationen ausgeführt werden: Wird eine Model Smell-Regel mit konkreten Simulink-Objekten aufgerufen, beweist oder widerlegt die Prolog-Engine die Regel für die Eingaben. Wird eine Regel mit Variablen aufgerufen, sucht die Prolog-Engine nach allen möglichen Kombinationen, die zur Erfüllung der Regel führen.

## 5 Weitere Beiträge

Zwei mögliche Model Smells wurden in diesem Paper präsentiert. Neben den beiden vorgestellten wurden dreizehn weitere Muster als Model Smells identifiziert und in Prolog implementiert. Mit der Prolog-Implementierung ist nach Generierung von Fakten zu einem Simulink-Modell die Suche nach Model Smells in diesem Modell möglich. Analysen haben allerdings gezeigt, dass die Einsetzbarkeit der Prolog-Implementierung in realen Modellen mit moderater Größe (mehr als 1000 Blöcke) unpraktikabel ist und die Laufzeit der Erkennung einiger Model Smells quadratisch mit der Anzahl der Blöcke wächst.

Daher wurde auf Basis der Prolog-Regeln ein Werkzeug in M-Script entwickelt, welches aufgrund der Verwendung der Simulink-API deutlich schneller arbeitet. Dieses Werkzeug bietet darüber hinaus einige besonders für Anwender interessante Funktionen:

Das Werkzeug bietet eine graphische Benutzeroberfläche, durch die gefundene Instanzen in einer Liste dargestellt werden. Zur schnellen Lokalisierung kann der Modellierer mittels Schaltflächen die zu einer Instanz gehörenden Modellteile farbig hervorheben.

## 6 Ausblick

Der Einsatz des Werkzeuges in einem realen Modell mit geringer interner Qualität hat gezeigt, dass die gefundenen Smell-Instanzen ein guter Ausgangspunkt zur zielgerichteten Verbesserung der Modellqualität sind. Ausbesserungsmaßnahmen zur Entfernung von Model Smells können mithilfe von werkzeuggestützten Transformationen [TD13] automatisiert durchgeführt werden.

Zur Weiterführung des Themas müssen weitere Model Smells insbesondere in Kooperation mit Simulink-Modellierern identifiziert, in einem Katalog aufgelistet und implementiert werden.

## Literatur


[Fow99] Martin Fowler. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, 1. Auflage, 1999.

[KKT11] Sören Kemmann, Thomas Kuhn und Mario Trapp. Extensible and automated model-evaluations with INProVE. In *Proceedings of the 6th international conference on System analysis and modeling: about models*, SAM'10, Seiten 193–208, Berlin, Heidelberg, 2011. Springer-Verlag.

[Sch12] Jan Scheible. *Automatisierte Qualitätsbewertung am Beispiel von MATLAB Simulink-Modellen in der Automobil-Domäne*. Dissertation, Universität Tübingen, 2012.

[TD13] Quang Minh Tran und Christian Dziobek. Ansatz zur Erstellung und Wartung von Simulink-Modellen durch den Einsatz von Transformationen/Refactorings und Generierungsoperationen. In Holger Giese, Michaela Huhn, Jan Phillips und Bernhard Schätz, Hrsg., *MBEES*, Seiten 1–12. fortiss GmbH, München, 2013.