Berechnung und Visualisierung der Modellkomplexität bei der mo- dellbasierten Entwicklung sicherheits....

Conference Paper · January 2010

CITATIONS
4

READS
99

3 authors, including:

Ingo Stürmer
Model Engineering Solutions Ltd.
41 PUBLICATIONS **311** CITATIONS
SEE PROFILE

Hartmut Pohlheim
Model Engineering Solutions GmbH
46 PUBLICATIONS **1,050** CITATIONS
SEE PROFILE

Berechnung und Visualisierung der Modellkomplexität bei der modellbasierten Entwicklung sicherheits-relevanter Software

Ingo Stürmer, Hartmut Pohlheim, Thorsten Rogier

Model Engineering Solutions GmbH, Friedrichstr. 50, D-10117 Berlin

Abstract

Die modellbasierte Entwicklung eingebetteter Software auf Basis von Simulink® und TargetLink® hat sich im Automobilbereich industriell etabliert, mittlerweile auch bei der Entwicklung sicherheits-relevanter Software-Funktionen. Zur Bewertung der Komplexität von Software ist es üblich Software-Metriken anzuwenden, wie z.B. Lines of Code, die Halstead-Metrik oder zyklomatische Komplexität. Diese Metriken lassen sich *nicht* direkt auf Modellebene übertragen und Komplexitätsmaße für Simulink und TargetLink sind noch unzureichend erforscht und wenig verbreitet. In diesem Papier präsentieren wir einen Ansatz, wie sich die Halstead-Metrik im Hinblick für Simulink- und TargetLink-Modelle adaptieren und gewichten lässt. Ziel ist es, ein Komplexitätsmaß zu entwickeln, mit dem sich komplexe Modellstrukturen identifizieren und visualisieren lassen, um so später Review- und Testaufwände abzuschätzen und die Entwicklung der Modell-Komplexität über den Entwicklungsprozess verfolgen zu können. Die Realisierung unseres Ansatzes wird an dem Werkzeug M-XRAY gezeigt, welches die genannten Ziele umsetzt.

1. Einleitung

Die modellbasierte Entwicklung eingebetteter Steuerungs- und Regelungssoftware für das Automobil ist durch den Einsatz ausführbarer Modelle in allen Entwicklungsphasen charakterisiert. Modellierungs- und Simulationswerkzeuge, die hier weite Verbreitung gefunden haben, sind die datenfluss-orientierte Modellierungssprache Simulink® und die zur Modellierung von Zustandsautomaten verfügbare Erweiterung Stateflow® von The MathWorks [2]. Codegeneratoren, wie *TargetLink* von dSPACE [3] oder der *Real-Time Workshop/Embedded Coder* von The MathWorks [2] erlauben es, automatisch effizienten C Code direkt aus diesen Modellen zu generieren. Dabei ist insbesondere die Qualität des generierten Codes aber auch dessen Komplexität entscheidend. Beides resultiert unmittelbar aus der Qualität und Komplexität der Modelle, die für die Codegenerierung verwendet werden. Daher ist es wichtig, qualitätssichernde Maßnahmen und Messungen der Komplexität möglichst früh im Entwicklungsprozess, d.h. bereits auf Modellebene durchzuführen. Die Erhebung der Modellkomplexität versteht sich dabei als ergänzende Maßnahme zur analytischen Qualitätssicherung der Modelle aber auch zur Überwachung des Entwicklungsprozesses. Die Schritte der Qualitätssicherung, die bei der modellbasierten Entwicklung eingebetteter Software im Automobil angewendet werden, sind in [4] erläutert und werden daher hier nicht weiter vertieft. Diese Arbeit betrachtet die Messung und Bewertung der Komplexität von Simulink und TargetLink Modellen. Bewertet werden hierbei die Verständlichkeit, Wartbarkeit, Fehleranfälligkeit, etc. der Modelle. Hintergrund dieser Arbeit war der Bedarf zur Messung und Visualisierung der Modellstruktur und –komplexität im Rahmen von Modell-Reviews, welche wir in zahlreichen Kundenprojekten als qualitätssichernde Maßnahme durchgeführt haben. Die internationale Norm ISO 26262 fordert beispielsweise die Beschränkung auf eine möglichst niedrige Komplexität für alle sicherheits-relevanten Software-Funktionen[1]. Die Komplexität der Software sollte dabei durch die Software-Architektur bestimmt werden (ISO 26262-6, Req. 6.6). Wie sich dies auf Modellebene anwenden lässt, vertiefen wir in einem gesonderten Abschnitt. Zunächst werden wir ein Beispielmodell aus

[1] Vgl. ISO 26262-6, Req. 4.4.7, Table 1

dem Automotive-Umfeld einführen und dabei einige Modellierungspattern erläutern, die wir später bei der Berechung der Modellkomplexität verwenden werden.

1.1 Beispiel eines Simulink Modells

In Abb. 1 betrachten wir ein gängiges Simulink Beispielmodell zur fehlertoleranten Kraftstoffregelung, *fuel rate controller*, das als Demo-Modell in allen gängigen Simulink-Installationen frei verfügbar ist. Das Modell realisiert die Regelung der Kraftstoffzufuhr unter der Berücksichtigung fehlerhafter Sensorsignale. Da die eigentliche Funktion des Modells hier nicht von Bedeutung ist, betrachten wir die Modelierungselemente und Struktur des Modells.

Das Modell in Abb. 1 wird durch einen Datenfluss von links nach rechts beschrieben. Auf der linken Seite werden Quellsignale (*throttle*, *engine speed*, etc.) über so genannte Eingabeports (Inports) zugeführt, die im oberen linken Teil dann zu einem Signalbus zusammengeführt. Anschließend wird dieser dem Subsystem *Sensor Correction and Fault Redundancy* als Signalbus *Sensors* zugeführt werden. Im linken unteren Teil befindet sich ein Zustandsautomat *control logic*, der auf Basis bestimmter Ereignisse die Signale *fail_state* und *fuel_mode* erzeugt. Das Signal *fail_state* wird als *Failures* in den oben genannten Subsystem und in *Fuel Calculation* (vgl. auch Abb. 8) weiter verarbeitet.

Ergebnis dieses Modells ist die Berechnung des Ausgabesignals (Outport) *fuel rate*, das die jeweilige Kraftstoffzufuhr vorgibt.

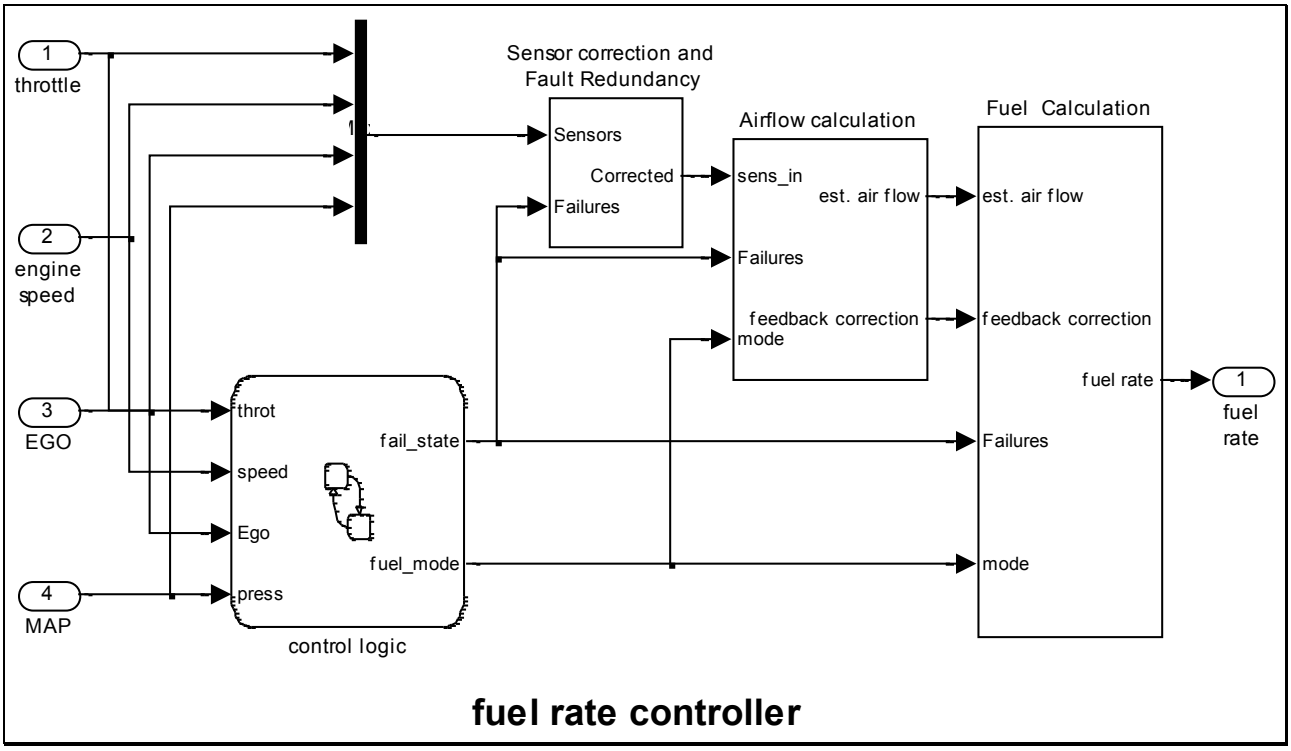

Abb. 1: Simulink Beispielmodell zur fehlertoleranten Kraftstoffregelung

Im Folgenden betrachten wir die Merkmale der Simulink-und TargetLink-Modellarchitektur, sowie deren Einfluss auf die Software-Architektur. Hierfür sollen zwei Begriffe eingeführt werden:

- **Modell-Architektur:** bezeichnet bei Simulink- und TargetLink-Modellen die Partitionierung der Modelle mit Hilfe von Subsystemen, die ebenso die hierarchische Tiefe des Modells vorge-

ben. Neuerdings kann dabei auch die Modularisierung der Modelle über das so genannte *Model Referencing*[2] hinzugezählt werden.

- **Software-Architektur:** bezeichnet hier die Aufteilung und Struktur des aus dem Modell generierten Codes. Die Struktur des Codes wird dabei explizit über das Modell und die Konfiguration des Codegenerators bestimmt. Wesentliche strukturierende Elemente sind atomare Subsysteme, die als (wieder verwendbare) Funktionen deklariert sind. Dies kann durch spezielle Blöcke (Function Block) oder eine spezielle Subsystem-Konfiguration erfolgen. Beim *Model Referencing* wird für jedes Modell (Modul) ein separates Programm generiert, das wieder nach der im Modell hinterlegten expliziten Subsystem-Konfiguration strukturiert ist.

Die Architektur des Simulink Modells wird maßgeblich durch die *Subsysteme* (rechteckige Kästen) bestimmt, die eine hierarchische Struktur vorgeben, da diese wiederum Blöcke und Subsysteme kapseln bzw. enthalten. Die in Abb. 1 gezeigten Subsysteme haben jedoch keinen Einfluss auf die Semantik des Modells, daher werden diese Subsysteme auch als *virtuell* bezeichnet. Dies unterscheidet sie von nicht-virtuellen, i.e. atomaren, Subsystemen, da letztere mit ihren enthaltenen Blöcken in einem Simulationsschritt ausgeführt werden. Dieser Hinweis ist wichtig, da atomare Subsysteme die Software-Architektur bei der Codegenerierung vorgeben. So werden beispielsweise (wieder verwendbare) Software-Funktionen im Modell mit Hilfe von atomaren Subsystemen, häufig als Bibliotheksblöcke, modelliert.

Es zeigt sich bereits auf dieser Betrachtungsebene, dass die Komplexität des gezeigten Modells sich aus der Anzahl und Anordnung der Blöcke, sowie deren „Verdrahtung“ der Ein- und Ausgänge ergibt. Mögliche Rückkoppelungen im Signalfluss erhöhen diese Komplexität, wie dies z.B. bei Reglern der Fall ist. Als zweite Dimension kommt die hierarchische Tiefe des Modells hinzu sowie der Querverweis auf andere Modellteile. Diese Punkte werden wir bei der Komplexitätsberechnung wieder aufgreifen (Abschnitt 2.1).

Ein vereinfachtes virtuelles Subsystem ist in Abb. 2 gezeigt. Hier wird ein Kontrollfluss in Simulink modelliert, der auf Basis der Eingangssignale *Value* und *MaxValue* dem Ausgangssignal *Result* entweder den Wert *5* oder *100* zuweist. Der jeweilige Wert (5 oder 100) wird über den mittleren Eingang am *Switch* Block bestimmt, mit dem der Kontrollfluss ähnlich einer *if-then-else* Struktur modelliert wird.

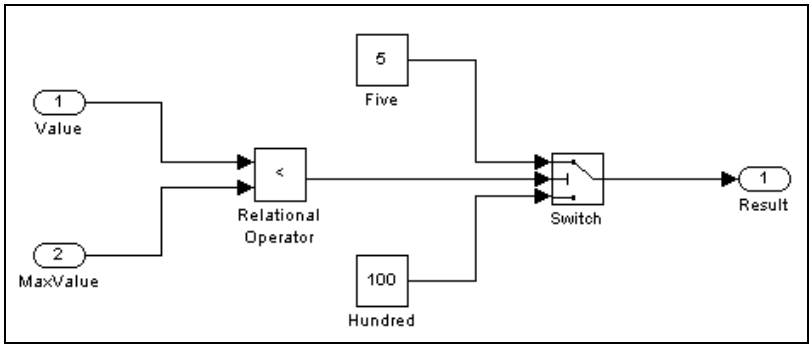

Abb. 2: Beispiel eines Simulink-Subsystems zur Modellierung eines Kontrollflusses

Wir werden dieses vereinfachte Beispiel bei der konkreten Berechung der Modellkomplexität wieder aufgreifen.

[2] Model Referencing: bezeichnet die Verwendung von *Model*-Blöcken anstatt der gewohnten Subsystem-Blöcke. Dabei werden, ähnlich der Verwendung von Library-Blöcken, ganze Modell-Dateien als Submodelle hierarchisch eingebunden. Bei der Codegenerierung werden diese wie einzelne Module bzw. Unterprogramme des Hauptmoduls behandelt.

1.2 Beispiel eines Stateflow-Charts

Ein Ausschnitt aus dem in Abb. 1 gezeigten Stateflow-Chart *control logic* ist in Abb. 3 widergegeben. Der Zustand *Pressure_Sensor_Mode* beinhaltet zwei Unterzustände, *press_norm* und *press_fail*, die über Transitionen aktiv bzw. inaktiv geschaltet werden können. So wechselt das System z.B. vom normalen Druck (*press_norm*) in den Zustand fehlerhaft (*press_fail*), wenn der Druck *press größer max_press* oder kleiner *min_press* ist. Gilt diese Bedingung, wir zusätzlich im externen Zustand *Sens_Failure_Counter* das Ereignis *INC* ausgelöst.

Die Berechnung der Komplexität dieses Chart wird später durchgeführt (Abschnitt 2.5.2).

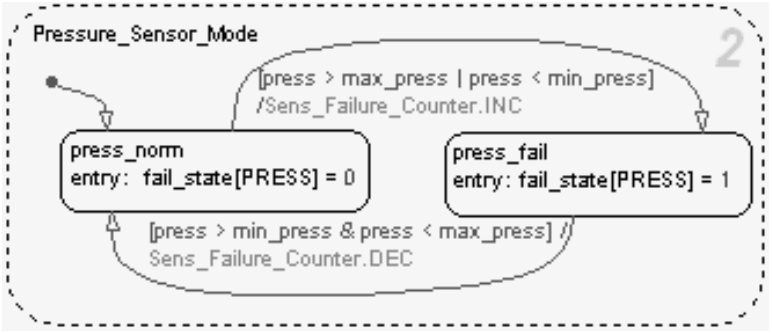

Abb. 3: Ausschnitt aus einem Stateflow Chart mit Zustand *Pressure_Sensor_Mode* und Unterzuständen *press_norm* und *press_fail*

Das Papier wird im Folgenden aufzeigen, wie sich Software-Komplexitätsmaße auf Simulink, Stateflow und TargetLink Modelle übertragen lassen und welche Erfahrungen wir im Rahmen von Modell-Reviews mit dem Werkzeug M-XRAY [7] gesammelt haben.

2. Berechnung der Modell-Komplexität

Software-Metriken bemessen Software-Einheiten wie z.B. Funktionen oder Module anhand von Maßzahlen. Diese bewerten die Komplexität der betrachteten Software-Einheit durch numerische Gewichtungen. Weit verbreitete Vertreter dieser Metriken sind beispielsweise:

1. **Lines of Code:** die Anzahl ausführbarer Programmzeilen
2. **Zyklomatische Komplexität:** die Anzahl der Pfade im Kontrollfluss des Programms
3. **Halstead Metrik:** die Bemessung der Volumens eines Programms anhand der Komplexität von Operationen und den zugehörigen Operanden (siehe Abschnitt 2.2)

Mit diesen Komplexitätsmaßen lassen sich beispielsweise unterschiedliche Entwicklungsstände und Versionen einer Software-Einheit miteinander vergleichen und die Zunahme oder Abnahme der Programmkomplexität betrachten. Häufig werden auch die Maße erhoben, um Prüf- und Review-Aufwände abzuschätzen – dies war auch unser Ziel.

2.1 Übertragung von Software-Metriken auf Modelle

Eine Metrik für Software lässt sich nicht direkt auf Simulink und TargetLink Modelle übertragen: aus diesem Grund gibt es vermutlich auch noch keine allgemein akzeptierten Maße, um die Komplexität der Modelle zu bewerten. Bei den Modellen wird häufig die Anzahl der verwendeten Blöcke und Blocktypen erhoben - vergleichbar mit dem Zählen der ausführbaren Programmzeilen (*Lines of Code*). Damit lässt sich die Größe eines Modelle zwar bemessen, aber die tatsächliche Komplexität nicht stichhaltig begründen. Die *Verification & Validation Toolbox* von Simulink (V&V Toolbox) [2] berechnet die Komplexität der Modelle anhand ihrer Zyklomatischen Komplexität (*engl.*: cyclomatic complexity, CC im Folgenden), d.h. anhand der Pfade des Kontrollflusses.

Zur Berechnung werden dabei Blöcke herangezogen, die für den Kontrollfluss relevant sind (Switch, MinMax, Saturation, etc.). Die Berechnung der Zyklomatischen Komplexität mit der V&V Toolbox ist nach unserer Erfahrung eine verbreitete Methode, um die Komplexität der Modelle zu berechnen.

Folgende Gründe sprechen aus unserer Sicht aber dagegen, die Zyklomatische Komplexität als entscheidendes Maß zur Bewertung der Modellkomplexität heranzuziehen: Die Komplexität der Modelle wird *nicht* maßgeblich durch den Kontrollfluss bestimmt, sondern durch die Anzahl der Blöcke und wie diese mit Signalen untereinander verschaltet sind.

Beispiel: in Abb. 8 (Abschnitt 3.2) ist das Subsystem *Airflow calculation* aus Abb. 1 gezeigt. Für dieses haben wir die Zyklomatische Komplexität mit der *V&V Toolbox* berechnet. Der errechnet Wert ist *1*, da nur ein Verzweigungspunkt im Kontrollfluss des gezeigten Subsystems vorhanden ist (Switch-Block *hold integrator*). Bei Subsystemen, die keinen Verzweigungspunkt enthalten, fließen diese bei der Betrachtung der Zyklomatischen Komplexität gar nicht in die Berechnung mit ein! Mit unserer Methode, dass Modell-Volumen über die Halstead Metrik zu berechnen, haben wir einen Wert von MV=*344* errechnet (mehr dazu in Abschnitt 2.2 und 2.4). Dieses Subsystem zeigt damit besonders deutlich, dass sich die Komplexität eines Simulink Subsystems nicht durch den Kontrollfluss, sondern durch den Signal- bzw. Datenfluss bestimmt.

Anders ist dies bei Stateflow: die Charts mit Ihren Zustandsübergängen beeinflussen deutlich die Zyklomatischen Komplexität eines Modells. So enthält beispielsweise der Chart *Control Logic* aus Abb. 1 fast 2/3 der Zyklomatischen Komplexität des gesamten Modells (V&V Toolbox: CC=50 der insgesamt CC=83 Punkte). Nach unser Berechnung sind es nur 1/3 der Gesamtkomplexität (MV=881 von 2338 Punkten). Die Zyklomatische Komplexität für den Zustand *Pressure_Sensor_Mode* beträgt immerhin nach der V&V Toolbox CC=5 (bei uns: MV=*100*). Stellt man diese *5* dem Subsystem *Airflow calculation* aus Abb. 8 mit einer Komplexität von CC=*1* gegenüber, zeigt sich hier ein deutliches Ungleichgewicht. Dahingegen scheint das Verhältnis MV=100 für den Chart zu MV=344 zu dem Subsystem realistischer.

Im Folgenden betrachten wir, welche Eigenschaften des Modells aus unserer Erfahrung relevant für die Komplexitätsbetrachtung sind:

1. **Schnittstellen:** Die Anzahl der Ein- und Ausgänge eines Subsystems. Dies ist entspricht der Betrachtung von Funktionsschnittstellen und ist relevant für die Test- und Integrierbarkeit der Subsysteme.
2. **Granularität:** Die Anzahl der auf einer Subsystem-Ebene liegende Blöcke. Vergleichbar ist dies mit der Länge des Programmcodes für eine Funktion.
3. **Verknüpfung:** Die Anzahl der jeweiligen Signal-Eingänge und Signal-Ausgänge der einzelnen Blöcke, sowie deren Verdrahtung. Die Komplexität der Verknüpfung der Elemente hat Einfluss darauf, wie gut die Modelle zu verstehen und zu warten sind.
4. **Rückkopplung:** Rückführungen im Signalfluss sind bei Algorithmen aus dem regelungstechnischen Umfeld üblich. Diesen tragen insbesondere zur Komplexität bei, wenn die Rückkopplung über andere hierarchische Subsystem-Ebenen erfolgt.
5. **Gewichtung:** Eine Bewertung der einzelnen Blöcke, die den Aufwand bemisst, der für die Prüfung des Blocks erforderlich ist.
6. **Verschachtelung:** die hierarchische Tiefe der Modelle, was vergleichbar dem Aufrufgraphen für Funktionen in der klassischen Programmierung ist.

2.2 Verwendung der Halstead Metrik für Modelle

Da wir die Komplexitätsmetriken maßgeblich benötigen, um Zeitaufwände für das Testen und den Review der Modelle abzuschätzen, lag es für uns nahe die Halstead Metriken für die modell-

basierte Entwicklung von Software auf Basis von Simulink und TargetLink zu übertragen, anzupassen und um Erfahrungswerte zu ergänzen.

Die Halstead-Metrik [1] stellt eine bewährte statische Analysemethode dar, um Komplexitätsmaße für Software quantitativ zu erheben, z.B. für C oder Java Code. Die Berechnung der Halstead-Metrik basiert auf der Annahme, dass ausführbare Programmteile einer Software aus *Operatoren* wie Additionen, Vergleichsoperatoren, etc. und *Operanden*, z.B. Konstanten und Variablen, aufgebaut sind. Die genaue Defintion der Operatoren und Operanden ist jedoch abhängig von der betrachteten Programmiersprache und vom jeweiligen Kontext. Daher ist es nicht unüblich, die Bedeutung der Begriffe Operatoren und Operanden kontextabhängig neu zu definieren.

Folgende Basismaße werden für die Berechnung der Halstead-Metrik für Programmcode betrachtet:

- **N1** : Anzahl der Operatoren (insgesamt)
- **N2** : Anzahl der Operanden (insgesamt)
- **N3** : Anzahl unterschiedlicher Operatoren
- **N4** : Anzahl unterschiedlicher Operanden

Von den verschiedenen Halstead-Metriken, die sich mit Hilfe dieser Basismaße berechnen lassen, betrachten wir im Folgenden das Halstead-Volumen (HV), das sich wie folgt berechnet:

Halstead-Volumen: `HV = (N1 + N2) * log2 (N3 + N4)`

Die Berechnung von HV sol an einem einfachen Pseudo-Code Beispiel verdeutlicht werden:

Beispiel:

```
if (Value < MaxValue){
     Value =5;
} else {
     Value = MaxValue;
}
```

Das Beispielprogramm berechnet den Wert *Value* in Abhängigkeit von den variablen Größen *Value* und *MaxValue*. Die Berechung der Basismaße und des *HV* ist in Tabelle 1 wieder gegeben:

Tabelle [1]: Berechnung der Basismaße und des Halstead-Volumens (HV) für Pseudo-Code Beispiel

	Operatoren				Operanden			
Basismaß	**If()**	**else**	**<**	**=**	**5**	**Value**	**MaxValue**	**Summe**
N1	1	1	1	2				**5**
N2					1	3	2	**6**
N3	1	1	1	1				**4**
N4					1	1	1	**3**

Aus der Tabelle 1 ergibt sich damit ein HV von:

`HV = (5 + 6) * log2 (4 + 3) = 30.8809`

Im nächsten Abschnitt betrachten wir, wie die Basismaße und das Halstead-Volumen auf Simulink-Modelle angewendet werden können.

2.3 Modell-Komplexität für Simulink und TargetLink

Bei der Berechnung der Modell-Komplexität wird nicht zwischen Simulink- und TargetLink-Blöcken bzw. -Modellelementen unterschieden, da TargetLink-Blöcke Simulink-Blöcke maskieren. Die TargetLink-Blöcke werden vom Entwickler bei der Modellierung mit Informationen für die Codegenerierung angereichert. Dies beinhaltet beispielsweise Festkomma-Skalierungen und TargetLink-Datentypen, die auf das TargetLink-spezifische Data Dictionary verweisen. Die TargetLink-Blöcke sind damit bei der Komplexitätsbetrachtung der Modelle zu Simulink auf graphischer Ebene vergleichbar. Die unterschiedlichen Konfigurationen der Blöcke sollten aus unserer Sicht für die Komplexitätsberechnung durch block-spezifische Gewichtungen berücksichtigt werden.

Die Anwendung der Halstead Metriken auf Simulink und TargetLink Modelle ist erst dann sinnvoll, wenn eine Anpassung im Hinblick auf die Bedeutung der Operatoren, sowie deren Ein- und Ausgänge vorgenommen wird. Unsere Erfahrung hat gezeigt, dass es bei graphischen Modellen wie Simulink oder Stateflow nicht zielführend ist, zwischen Operatoren, z.B. Additionsblöcke, und Operanden, z.B. Konstanten und Signale, für die Berechnung der Komplexität zu unterscheiden. Jeder Block, ob funktional oder nicht, ist wie ein Operator und seine Signal-Eingänge und Signal-Ausgänge sind als Operanden zu betrachten, da die Modellkomplexität abhängig vom Grad der Verknüpfung der Blöcke ist. Diese Aussage wird auch in [6] gestützt. Wir berücksichtigen jedoch, dass die Blocktypen eine unterschiedliche Komplexität aufweisen. So ist ein Additionsblock komplexer als eine ODER-Verknüpfung, da hier ggf. unterschiedliche Datentypen der Operanden (Signale) berücksichtigt werden müssen. Diesen Unterschied tragen wir durch eine individuelle, adaptierbare Gewichtung der einzelnen Blöcke Rechnung. Das Maß der Gewichtung haben wir in zahlreichen Modellreviews für jeden Blocktyp erarbeitet[3]. Die entwickelten Gewichtungen wurden dann später mit den Aufwänden bereits durchgeführter Reviews abgeglichen und angepasst.

2.4 Berechnung des Modell-Volumens für Simulink und TargetLink

Im Folgenden betrachten wir die Definition der adaptierten Halstead-Metriken für Simulink Subsysteme zur Berechnung der Modell-Komplexität (Modell-Volumen):

- N1 : Anzahl der Blöcke
- N2 : Anzahl der Eingänge eines Blocks
- N3 : Anzahl unterschiedlicher Blocktypen
- N4 :Anzahl der Ausgänge eines Blocks

Die oben genannten Maßzahlen beziehen sich jeweils auf ein betrachtetes Subsystem. Das von uns neu definierte Maß zur Berechung der Modellkomplexität bezeichnen wir als *Modell-Volumen MV*, das sich wie folgt berechnet:

Modell-Volumen `MV= (N1 + N2) * log2 (N3 + N4 + 1).`

Damit ergibt für das in Abb. 2 gezeigte Subsystem folgende Berechnung des Modell-Volumens, die in Tabelle 2 gezeigt ist. Eine Gewichtung und logarithmische Reduzierung der Blöcke wurde bereits durchgeführt.

Das Modell-Volumen berechnet sich wie folgt:

`MV= (N1 + N2) * log2(N3 + N4 + 1) = (6.9 + 7.2) * log2(5.6 + 6.1 + 1) = 51.7`

Unabhängig von der Betrachtung einzelner Blöcke werden Rückkopplungen im Signalfluss gesondert berechnet.

[3] Unbekannte Blöcke erhalten den default-Wert 1.

Tabelle 2: Basismaße für Simulink Beispiel aus Abb. 2 mit Gewichtung und logarithmischer Reduzierung der Komplexität der Ein- und Ausgangswerte

Weight	0.8	0.8	1.5	0.5	0.5	2	0.8	
Basismaß	Inport value	Inport MaxValue	<	5	100	Switch	Outport result	Result
N1	0.8	0.8	1.5	0.5	0.5	2	0.8	6.9
N2	0	0	2.4	0	0	4	0.8	7.2
N3	0.8		1.5	0.5		2	0.8	5.6
N4	0.8	0.8	1.5	0.5	0.5	2	0	6.1

2.5 Modell-Komplexität für Stateflow

Da ein Stateflow-Chart aus anderen graphischen Elementen besteht als ein Simulink Subsystem (Zustände, mit Bedingungen annotierte Transitionen, Actions, Events, etc.) haben wir eine andere Methode gewählt, um die Komplexität der Charts zu berechnen.

Der Algorithmus zur Berechnung der Stateflow-Metriken basiert zunächst auf der Anzahl der Zustände, Actions, Events und der Anzahl und Scope von Bedingungen.

2.5.1 Komplexitätsberechnung für Zustände

Die Komplexität eines Zustands berechnet sich mit Hilfe der Basisgewichte für die Zustandsobjekte wie folgt:

- **+1** für den Zustand selbst
- **+1** für Entry-Actions
- **+1** für During-Actions
- **+1** für Exit-Actions
- **+1** für OnEvents
- + Maßzahl für die Komplexität der enthaltenen Transitionen (werden gesondert berechnet)

Es sollte dabei möglich sein, die Basisgewichte projektspezifisch anzupassen.

2.5.2 Komplexitätsberechnung für Transitionen

Transitionen erhöhen die Komplexität eines Zustands der diese enthält. Wir berücksichtigen dabei die folgenden Stateflow Objekte: Conditions (Bedingungen), Condition Actions, Transition Actions und Events.

Die Komplexität eines Zustands resultiert damit aus der Summe seiner Basisgewichte:

- **+0** für Transitionen die mit keiner Condition oder Condition Actions assoziiert sind
- **+4** für Transitionen mit einer Bedingung
- **+2** für jeden atomaren Ausdruck einer zusammengesetzten Bedingung
- **+1** für jede Action die mit einer Condition verknüpft ist
- **+1.5** für jede Action die mit einer Transition verknüpft ist
- **+4** für jedes Event

Beispiel:

Die Berechnung der Komplexität eines Zustands wollen wir anhand des Zustands *Pressure_Sensor_Mode* in Abb. 3 zeigen.

Die lokale Komplexität der zwei Unterzustände *press_norm* und *press_fail* ist jeweils 2: ein Basispunkt für den Unterzustand selbst und einen für die Entry-Action. Die lokale Komplexität von *Pressure_Sensor_Mode* selbst (+1) wird durch die zwei Transitionen erhöht, welche die Unterzustände *press_norm* und *press_fail* verknüpfen. Zusätzlich erhalten diese +4 Basispunkte für die Bedingung und weitere +2 Punkte für die zusammengesetzten, atomaren Ausdrücke. Zusätzlich werden +1.5 Punkte für die Transition-Actions addiert.

3. Visualisierung der Modellstruktur und der Modellkomplexität

Nach der Erhebung der Modellkomplexität kommt der Darstellung der Ergebnisse eine große Bedeutung zu. Diese müssen so umfassend wie nötig und gleichzeitig kompakt dargestellt werden. Dafür betrachten wir im nächsten Abschnitt zwei Verfahren, um die Modellstruktur und die Modell-Komplexität möglichst kompakt zu vermitteln. Anschließend werden wir die Unterschiede zwischen lokaler und globaler Modell-Komplexität erläutern, die insbesondere für die Architektur und Struktur der Modelle von Bedeutung ist.

Path	Name	Level	Info	Local Complexity	Global Complexity
-	fuelsys	0	(Root)	**96**	2338
fuelsys	engine gas dynamics	1	-	**33**	352
fuelsys/engine gas dynamics	Mixing & Combustion	2	-	**59**	59
fuelsys/engine gas dynamics	Throttle & Manifold	2	-	**42**	260
fuelsys/engine gas dynamics/Throttle & Manifold	Intake Manifold	3	-	**67**	67
fuelsys/engine gas dynamics/Throttle & Manifold	Throttle	3	-	**151**	151
fuelsys	fuel rate controller	1	-	**58**	1890
fuelsys/fuel rate controller	Airflow calculation	2	-	**344**	344
fuelsys/fuel rate controller	Fuel Calculation	2	-	**75**	275
fuelsys/fuel rate controller/Fuel Calculation	Switchable Compensation	3	-	**200**	200
fuelsys/fuel rate controller	Sensor correction and Fault Redundancy	2	-	**134**	332
fuelsys/fuel rate controller/Sensor correction and Fault Redundancy	MAP Estimate	3	-	**66**	66
fuelsys/fuel rate controller/Sensor correction and Fault Redundancy	Speed Estimate	3	-	**66**	66
fuelsys/fuel rate controller/Sensor correction and Fault Redundancy	Throttle Estimate	3	-	**66**	66
fuelsys/fuel rate controller	control logic	2	Chart	11	881
fuelsys/fuel rate controller/control logic	Oxygen_Sensor_Mode	3	State	80	110
fuelsys/fuel rate controller/control logic/Oxygen_Sensor_Mode	O2_fail	4	State	10	10
fuelsys/fuel rate controller/control logic/Oxygen_Sensor_Mode	O2_warmup	4	State	10	10
fuelsys/fuel rate controller/control logic/Oxygen_Sensor_Mode	O2_normal	4	State	10	10
fuelsys/fuel rate controller/control logic	Pressure_Sensor_Mode	3	State	80	100
fuelsys/fuel rate controller/control logic/Pressure_Sensor_Mode	press_norm	4	State	10	10
fuelsys/fuel rate controller/control logic/Pressure_Sensor_Mode	press_fail	4	State	10	10

Abb. 4: Tabellarisch verkürzte Visualisierung von Modellstruktur und zugehöriger lokaler und globaler Modellkomplexität

3.1 Analyse und Visualisierung der Modellstruktur

Die Granularität der Darstellung sollte sich auf der einen Seite an der Architektur des Modells orientieren. Dafür kann die Struktur des Modells als Baum von Subsystemen und Blöcken verwendet werden. Für diese Darstellung der Modellstruktur stellen wir zwei Varianten vor, die sich in unserer Arbeit bewährt haben.

- Tabelle mit Pfad, Name und Tiefe/Ebene des Subsystems (siehe Spalte 1-3 in Abb. 4) und
- Hierarchische Strukturdarstellung nur mit Subsystemnamen (siehe Abb. 5).

3.1.1 Tabellarische Darstellung der Modellstruktur

Bei der tabellarischen Darstellung in Abb. 4 ist die Struktur des Modells durch die Pfadangabe in der ersten Spalte *Path* nachvollziehbar. Durch die Tennung von Pfad und Block- bzw. Subsystem-Namen sieht der Modellierer, welche Subsysteme zu betrachten sind. Der *Level* (3. Spalte) gibt zusätzlich an, in welcher hierarchischen Tiefe des Modells sich das entsprechende Subsystem befindet: gerade bei großen Modellen mit 15 und mehr Ebenen eine wichtige Orientierung. Ob es sich bei dem "Subsystem" um ein Stateflow-Chart oder -Zustand handelt, wird in der 4. Spalte dargestellt (*Chart* bzw. *State* in Spalte 4: *Info*). Für eine schnelle Navigation ins Modell sind die Namen von Pfad bzw. Name immer auf die entsprechenden Subsysteme im Modell verlinkt.

3.1.2 Hierarchische Darstellung der Modellstruktur

Eine zweite Möglichkeit der Darstellung der Modellstruktur stellt die hierarchische Visualisierung dar. Wie in Abb. 5 zu sehen, wird dazu die Baumstruktur mit den Namen der Subsysteme und ihrem Modell-Volumen (*Comp*) gezeigt. Die dargestellte Informationsmenge zur Modellstruktur verringert sich dadurch wie folgt:

- Der Pfad des Subsystems bzw. Blocks muss nicht mehr angezeigt werden
- Mehrfach verwendete Subsysteme bzw. Bibliotheksblöcke werden nur bis zu Ihrem Referenzpunkt angezeigt, die darin enthaltene Modellstruktur wird nur einmal visualisiert (mehr dazu unten)

Die hierarchische Tiefe eines Subsystems ergibt sich indirekt aus der Darstellung. Bei großen Modellen stößt aber auch diese Art der Darstellung aus Platzgründen an ihre Grenzen. In der Praxis ist dies aber ein geringes Problem, da erfahrene Modellierer eine niedrige Modellkomplexität und damit auch eine flache Modellhierarchie anstreben (idealerweise: < 10 Ebenen).

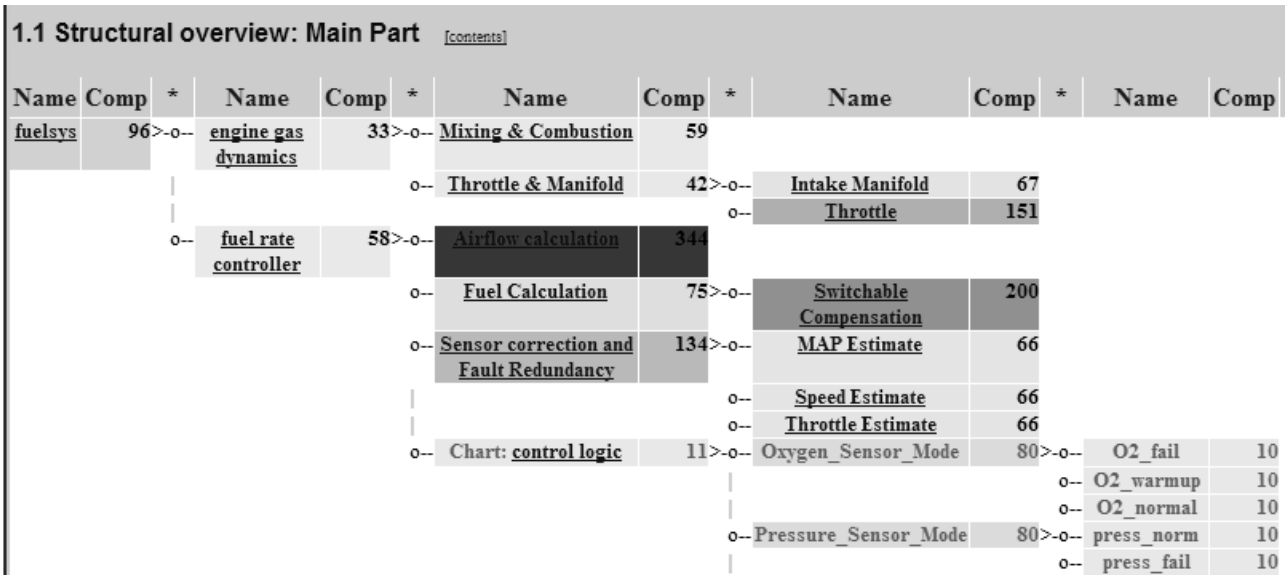

Abb. 5: Hierarchische Darstellung von Modellstruktur und lokaler Modellkomplexität

Bei der Darstellung der Modellstruktur fehlen noch zwei Aspekte – Bibliotheken (Libraries) und referenzierte Modelle (Model Referencing). Beide Bereiche können nach derselben Methode behandelt werden, daher stellen wir es hier nur die Vorgehensweise für Libraries dar.

Bibliotheken sind eine Variante, wiederverwendbare Blöcke/Subsysteme aus dem Modell zu verlagern und durch Links einzubinden. Oftmals werden Bibliotheken verwendet, um den Wartungsaufwand, Hierarchie und Größe des Modells zu reduzieren. Durch die Wiederverwendung muß die Struktur von Bibliotheksblöcken nicht an jeder verwendeten Stelle in der Modellstruktur angezeigt

werden. Nach unserer Erfahrung reicht es, die Struktur von Bibliotheksbestandteilen einmalig zu zeigen.[4] In der Modellstruktur wird nur auf diese separate Struktur verwiesen (nicht gezeigt in Abb. 5). Dies entspricht auch der Intention von Bibliotheken.

3.2 Unterscheidung zwischen lokaler und globaler Modellkomplexität

Die Komplexität eines Modells wird auch durch seine Hierarchie und die Granularität der einzelnen Subsysteme bestimmt. Aus diesem Grund ist es sinnvoll, zwischen lokaler und globaler Modell-Komplexität zu unterscheiden.

- **Lokale Komplexität:** umfasst die Blöcke und Verbindungen, die innerhalb eines Subsystems und nur auf der aktuellen (sichtbaren) Ebene liegen. D.h. die lokale Komplexität bezeichnet den Scope einer Metrik für die Modellelemente, die beim Blick auf ein Subsystem direkt zu sehen sind (Schnittstelle, Granularität, Verknüpfung, Rückkopplung). Die lokale Komplexität eignet sich nach unserer Erfahrung, um Aussagen über die Wartbarkeit zu treffen und um Review-Aufwände abzuschätzen.
- **Globale Komplexität:** diese umfasst auch alle unterhalb der aktuellen Ebene liegenden Blöcke und Verbindungen. In die globale Komplexität fließt damit die Komplexität der Bestandteile eines Subsystems komplett ein – also auch die Hierarchie. Daher kann die globale Komplexität auch als *hierarchische Komplexität* bezeichnet werden. Diese eignet sich nach unserer Erfahrung, um Aussagen über die Testbarkeit zu treffen und um Test-Aufwände abzuschätzen.

In Abb. 6 wird dieser unterschiedliche Scope von lokaler und globaler Komplexität graphisch veranschaulicht.

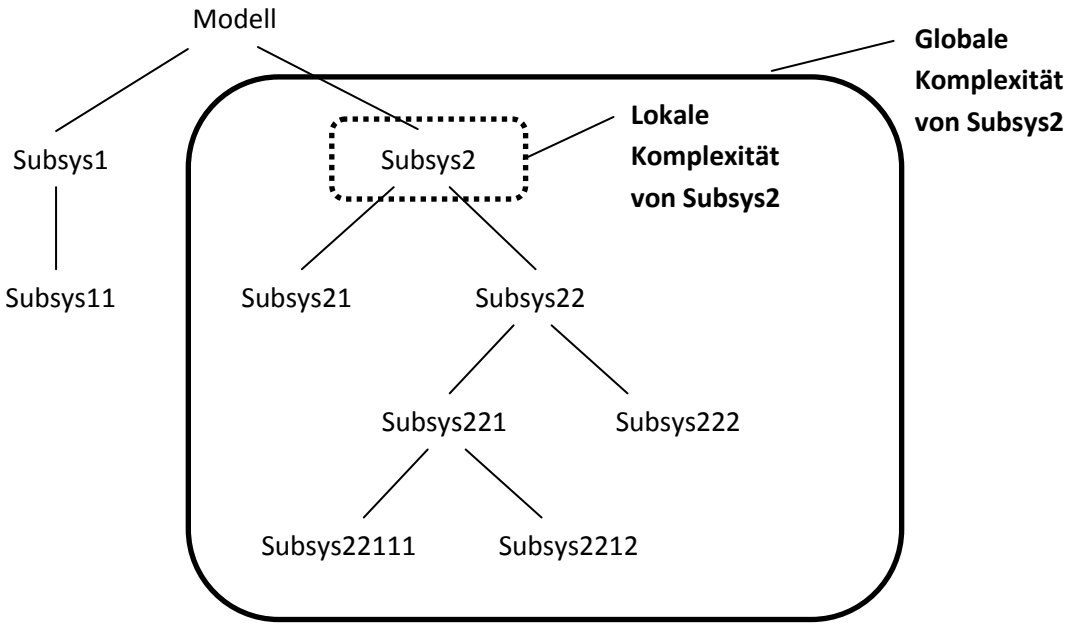

Abb. 6: Veranschaulichung von lokaler und globaler Modell-Komplexität

Die globale bzw. hierarchische Komplexität eines Subsystems ist damit die Summe aus seiner eigenen lokalen Komplexität und der aller direkt und indirekt, i.e. verlinkt, enthaltenen Subsysteme. Auf unterster Ebene (hier z.B. *Subsys22111*) sind lokale und globale Komplexität identisch.

[4] Dabei ist zu beachten, dass die Komplexität von Bibliotheken nur dann einmalig berechnet werden darf, wenn folgende Vorbedingungen gelten: die Bibliothek ist bereits konform mit gängigen Modellierungsrichtlinien; getestet; und die Komplexität wurde bereits gemessen und für angemessen bewertet.

Die lokale Komplexität bewertet den Teil eines Modellbereiches, der für den Modellierer sichtbar ist; die globale Komplexität bewertet auch alle darunter gelegenen "unsichtbaren" Teile dieses Modellbereiches.

Beispiele:

Als Beispiele sind in Abb. 7 das Subsystem mit einer niedrigen lokalen Komplexität und in Abb. 8 ein Subsystem mit der höchsten lokalen Komplexität aus dem Modell *fuelsys* (vgl. Abb. 1) dargestellt.

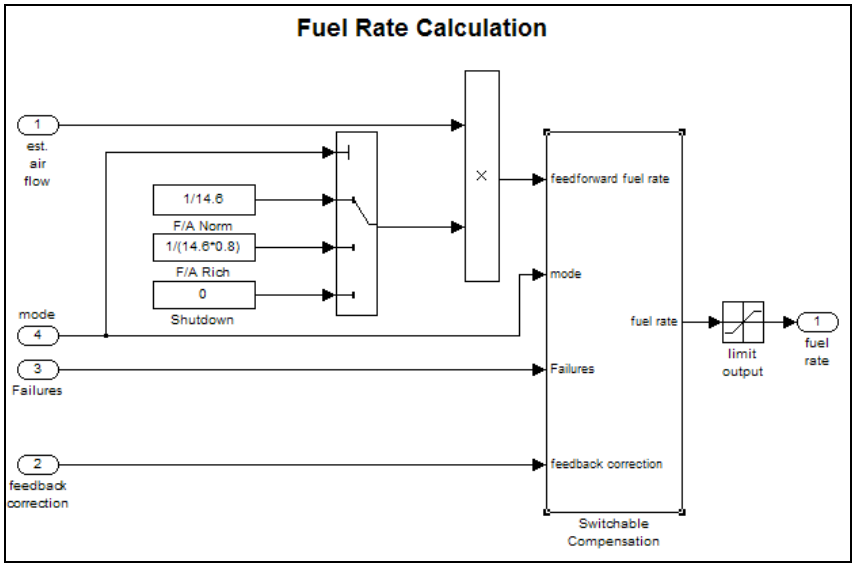

Abb. 7: Subsystem *Fuel Calculation* aus Abb. 1 mit niedriger lokaler Komplexität (MV=58)

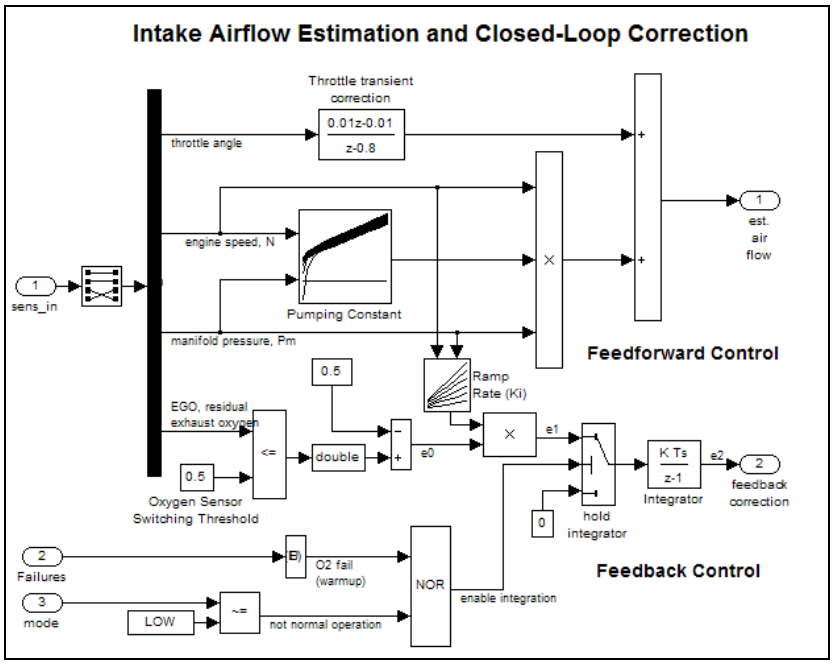

Abb. 8: Subsystem *Airflow calculation* aus Abb. 1 mit höchster lokaler Komplexität (MV= 344)

4. Lessons Learned

In zahlreichen Kundenprojekten haben wir sowohl im Pkw- als auch im Truck-Bereich aus unterschiedlichen Anwendungskontexten (Assistenzsysteme, Komfortfunktionen, Motorsteuerung, etc.) Modellreviews manuell und werkzeuggestützt durchgeführt. Schwerpunkte waren hierbei die Prüfung des Modells im Hinblick auf (a) korrekte Umsetzung der textuellen Anforderungen in die gra-

phische Modellierungssprache Simulink bzw. TargetLink, (b) Einhaltung von Modellierungsrichtlinien, (c) Bewertung der Modell- und daraus resultierenden Software-Architektur, (d) Robustheit der Arithmetik, (e) Optimierungspotentiale für die Generierung effizienten Seriencodes, etc. Dabei sind die Abschätzung des potentiellen Reviewaufwands sowie die Qualität des Modells von entscheidender Bedeutung.

Folgende Erfahrungen im Rahmen der Modellreviews und Anwendung unseres Ansatzes zur Komplexitätsberechnung von Modellen möchten wir hier wiedergeben:

- Die berechneten Komplexitätswerte zum Modell-Volumen decken sich mit den Erwartungswerten, die auf Basis von Peer-Reviews und Expertenwissen geschätzt werden.
- Die einzelnen Blöcke im Modell haben eine unterschiedliche Komplexität, unabhängig davon, ob man dies aus funktionaler Sicht, als Reviewaufwand oder Testaufwand betrachtet. Genau diese unterschiedliche Komplexität muss sich in der Metrik widerspiegeln (Gewichtung für die einzelnen Blocktypen).
- Mit der Anzahl der Ein- und Ausgänge eines Blocks steigt die Komplexität *nicht* linear an – daher wird die Anzahl der Ein- und Ausgänge bei der Komplexitätsberechnung logarithmisch bewertet.
- Wenn ein Block aus einer eigenen Library verwendet wird, so wird er auf jeder Ebene als Block/Subsystem gewertet. Die Komplexität innerhalb dieses Blocks wird aber nur einmal modelliert, getestet bzw. gereviewt. Also sollte diese Komplexität nur einmal in die globale Komplexität mit einfließen. Aus der Wiederverwendung dieses Blocks kann damit sogar eine "gesparte Komplexität" berechnet werden. Allerdings ist dieser Punkt bei uns als Parameter implementiert, da dies noch keine untermauerte Aussage ist.
- Wenn sich komplexe Modellstrukturen in sehr tiefen Ebenen "verstecken", sollte dies auch Anlass für eine Überarbeitung der Modell-Architektur sein. Diese Teile können über Links (Library) oder Modell-Referencing aus dem Modell verlagert werden. Dies verändert zwar nicht die Gesamtkomplexität, es zeigt aber die Bedeutung bzw. Größe dieser Teile und lässt sie nicht in der Tiefe des Modells untergehen.
- Die Zahlen der Modellkomplexität (lokal wie global) selbst sind zunächst dimensionslos. Sie bekommen ihre Bedeutung erst im Vergleich zu anderen (bekannten) Modellen bzw. im Vergleich zu über die Zeit erarbeiteten Schwellwerten.
- Da Schwellwerte sich schwer bestimmen lassen, haben wir Orientierungswerte erarbeitet, sich aus unseren Erfahrungswerten speisen.
- Die Komplexität der Modelle lässt sich deutlich reduzieren, wenn Modellierungsrichtlinien konsequent eingehalten werden.

5. Related Work

Sie Simulink V&V Toolbox [2] verfügt bereits über einen Mechanismus, um die Komplexität von Simulink-Modellen auf Basis der zyklomatischen Komplexität zu berechnen. Den Zusammenhang zwischen zyklomatischer Komplexität auf Modell- und auf Codebene für Simulink bzw. TargetLink-Modelle wurde erstmals in [5] untersucht. In [6] verwenden die Autoren ebenfalls die Simulink Blocktypen zur Berechnung der Komplexität, da sich die Modellkomplexität durch die Verknüpfung der Blöcke untereinander ergibt. Der Algorithmus orientiert sich hierbei auch an der Halstead-Metrik. Dieser wird zur Abschätzung der Personentage für Autocoding und Testen verwendet.

Wie wir sehen auch die Autoren von [6], dass die Anzahl der Knoten, die über die zyklomatische Komplexität berechnet wird, sich nur bedingt eignet, um Aufwände z.B. für das Testen abzuschätzen, jedoch um verschiedenen Entwicklungsstände eines Modells zu vergleichen.

6. Zusammenfassung

Mit der vorliegenden Arbeit konnte aufgezeigt werden, wie sich mit Hilfe eines praxis-orientierten Ansatzes Software-Metriken auf Simulink- und Stateflow-Modelle übertragen, anpassen und um Erfahrungswerte (Gewichtungen) zur Komplexitätsberechnung dieser Modelle ergänzen lassen. Die Berechnung und Visualisierung der Werkzeuge wurde im Werkzeug M-XRAY [7] realisiert. Hier hat sich die Halsted Metrik als geeignetes Maß gezeigt, um die für die Komplexitätsbetrachtung relevanten Eigenschaften zu berücksichtigen (Schnittstelle, Granularität, Verknüpfung, etc.). Die damit berechnete Metrik des Modell-Volumens eignet sich, um komplexe Modellstrukturen zu identifizieren und über eine Strukturbild des Modells zu visualisieren. Die Metrik des Modell-Volumens bleibt dabei jedoch dimensionslos und es lassen sich derzeit keine klaren Grenzwerte ableiten, wann ein bestimmter Schwellwert erreicht ist bzw. wie dieser festzulegen ist: der derzeitige Ansatz basiert auf Erfahrungswerten und Expertenwissen. Für beide Methoden wird angeregt, die wissenschaftliche Fundierung auszubauen.

In einer folgenden Veröffentlichung werden wir aufzeigen, wie sich die gewonnenen Komplexitätsmaße zur Abschätzung von Review-und Testauswänden anwenden lassen und wie sich die berechneten Maße mit den tatsächlichen Aufwänden decken.

7. Literatur

[1] Halstead, M.H.,: Elements of software science. Elsevier, New York, ISBN 0-444-00205-7 (Operating and programming systems series; Vol. 2, 1977.

[2] MathWorks, (product information) http://www.mathworks.com/products, 2010.

[3] dSPACE: TargetLink – Production Code Generator at http://www.dspace.com, 2010.

[4] *Fey, I., and Stürmer, I.*: Quality Assurance Methods for Model-based Development: A Survey and Assessment. SAE World Congress, SAE Doc. #2007-01-0506, Detroit, 2007. Also appeared in SAE 2007 Transactions Journal of Passenger Cars: Mechanical Systems, V116-6, Aug. 2008.

[5] Stürmer, I., Conrad, M., Fey, I., and Dörr, H.: Experiences with Model and Autocode Reviews in Model-based Software Development. Proc. of 3rd Intl. ICSE Workshop on Software Engineering for Automotive Systems (SEAS 2006), Shanghai, 2006.

[6] Vitkin, L; Dong, S.; Searcy, R.: Effort Estimation in Model-Based Software Development, World Congress, SAE Doc. #2006-01-0309, Detroit, 2006.

[7] Model Engineering Solutions GmbH, M-XRAY User Guide V1.2, 2010.

Danksagung

Die Arbeit entstand im Rahmen des Forschungsprojekts MQAC (Model Quality Assessment Center), gefördert durch das Bundesministerium für Wirtschaft und Technologie, Förderkennzeichen IW090031