Ein Qualitätsmodell zur automatisierten Ermittlung der Modellqualität bei eingebetteten Systemen

Jan Scheible, Ingo Kreuz
Daimler AG - Group Research and Advanced Engineering
{jan.scheible | ingo.kreuz}@daimler.com

Abstract: Die Qualitätssicherung der in der Automobilindustrie verwendeten Matlab Simulink-Modelle zur Softwareentwicklung von eingebetteten Systemen wird immer aufwendiger. Der Aufwand steigt, da immer mehr Funktionen im Fahrzeug als Software umgesetzt werden und der Umfang der Modelle stetig zunimmt. Diese Arbeit stellt einen Ansatz vor, der die Modellqualität mit Hilfe eines Qualitätsmodells und Metriken automatisiert ermittelt. Dadurch wird der Umgang mit sehr großen und komplexen Modellen möglich.

1 Motivation

Durch die modellbasierte Softwareentwicklung eingebetteter Systeme kann man im Vergleich zur Entwicklung mit textuellen Programmiersprachen unter anderem eine höhere Abstraktion und letztendlich eine höhere Softwarequalität erreichen [SVEH07]. Die höhere Abstraktion kann die Software-Entwicklung vor allem bezüglich nicht-funktionaler Qualitätsmerkmale unterstützen, weil Strukturiertheit, einheitliche Architektur, Wiederverwendbarkeit, Lesbarkeit und Übersichtlichkeit durch die grafische Notation unterstützt werden. Leider gibt die grafische Notation allein aber noch keine Garantie dafür, dass wirklich hohe Softwarequalität erreicht wird. So kann z. B. trotz grafischer Repräsentation eine ungünstige Architektur gewählt werden, Subsysteme falsch aufgeteilt werden oder Blöcke gewählt werden, die bei der Codegenerierung zu ineffizientem Code führen.

Modelle stellen bei der modellbasierten Entwicklung das zentrale Artefakt dar. Das bedeutet, dass die Modellqualität direkten Einfluss auf die Softwarequalität hat [FHR08]. Das heißt aber auch, dass durch die Qualitätssicherung von Modellen eine frühzeitige Absicherung erreicht werden kann: Fehler werden bereits im frühen Entwicklungsstadium erkannt und können noch kostengünstig behoben werden [FLS01]. Außerdem ist es im Automobilbau sehr wichtig, die Qualität eines Softwarestandes zu kennen, bevor er als Seriencode freigegeben werden kann, da es sich um eine sicherheitsrelevante Domäne handelt. Große Modelle in unserem Umfeld haben derzeit bis zu 15.000 Blöcke, 700 Subsysteme und 16 Hierarchieebenen.

Um die Qualität dieser Modelle mit vertretbarem Aufwand bewerten zu können, haben wir in [Sch10] ein Framework zur automatisierten Ermittlung der Modellqualität vorgestellt. In der aktuellen Arbeit gehen wir im Detail auf unser *Qualitätsmodell* ein, welches im Vergleich zu den klassischen Softwarequalitätsmodellen erweitert wurde. Dann beschreiben

wir die *Erhebung der Messwerte* durch die Metriken des Qualitätsmodells, welche nicht auf dem Quellcode, sondern auf den Modellen arbeiten. Dazu wird der schematische Ablauf der Erhebung und der Zugriff auf die Modelle beschrieben. Abschließend gehen wir auf die *Auswertung der Messergebnisse* ein.

2 Qualitätsmodell

Unser Ansatz für ein Qualitätsmodell folgt der Idee von Cavano und McCall, die bereits Ende der 70er Jahre das FCM-Qualitätsmodell (*Factors*, *Critera*, *Metrics*) für die Bewertung der Softwarequalität vorstellten [CM78]. Wir passen dieses Qualitätsmodell derzeit so an, dass es für grafische Modelle angewendet werden kann. Unser Fokus liegt dabei auf der Anwendbarkeit für Matlab Simulink-, Stateflow- und Targetlink-Modelle in beliebigen Entwicklungsphasen (im Folgenden *Simulink-Modelle* genannt), die in der Automobilindustrie besonders häufig zur Codegenerierung für eingebettete Systeme zum Einsatz kommen. Abbildung 1 zeigt einen Überblick über den momentanen Stand unseres Qualitätsmodells.

Bei Cavano und McCall setzt sich die Softwarequalität aus *Qualitätsfaktoren* (Factors) zusammen, die zum Teil allgemeingültig, aber zum Teil auch projektspezifisch sind. Wir haben diese Faktoren z. B. um den Qualitätsfaktor *Codegenerierbarkeit* erweitert, der für grafische Modelle hinzukommt. Außerdem wurden einige der ursprünglichen Faktoren zusammengefasst, um unser Qualitätsmodell übersichtlich zu halten. Dies war möglich, da manche Kriterien mehreren Faktoren zugeordnet werden konnten. Die Faktoren wiederum setzen sich aus *Qualitätskriterien* (Criteria) zusammen, welche die Faktoren konkretisieren. So wird z. B. der Qualitätsfaktor *Verständlichkeit* durch die Kriterien *Lesbarkeit*, *sprechende Bezeichner*, *kognitive Erfassbarkeit* u. Ä. konkretisiert. Kriterien können durch eine oder mehrere *Metriken* (Metrics) gemessen werden. Im Beispiel kann das Kriterium *Lesbarkeit* durch eine Metrik zur Messung der Schachtelungstiefe und der Anzahl der Rückkopplungen innerhalb eines Subsystems gemessen werden. Zur Qualitätsbewertung wird für jede Metrik neben einem Intervall mit erlaubten Werten eine Priorität definiert.

Zwar handelt es sich bei dem FCM-Modell um ein relativ altes und einfach strukturiertes Qualitätsmodell, aber unser Interesse gilt zunächst vor allem der Sammlung von Faktoren, Kriterien und Metriken, die spezifisch für die modellbasierte Entwicklung sind. So sind die meisten Metriken in unserem FCM-Modell nicht Simulink-spezifisch und daher auf andere Modellierungswerkzeuge übertragbar.

3 Erhebung der Messwerte

Dieser Abschnitt beschreibt die prototypische Anwendung unseres Qualitätmodells. Im Gegensatz zu Codemetriken messen die Metriken in unserem Ansatz direkt auf den Simulink-Modellen (vergleiche [Rau01]).

Abbildung 2 zeigt den schematischen Ablauf der Erhebung der Messwerte. Das Messobjekt

Abbildung 1: Momentaner Stand des Qualitätsmodells zur Modellqualitätsbewertung

Faktoren	Kriterien	Metriken	Beschreibungen
Codegenerierbarkeit	Verständlichkeit des Quellcodes	#Globaler Variablen	Wie viele globale Variablen gibt es im generierten C-Code?
		#Targetlink-Funktionen	Wie viele Subsysteme werden bei der Codegenerierung in C-Funktionen umgesetzt?
	Erhaltung der Architektur	#Architekturverletzungen	Ist die geplante Architektur im C-Code noch erkennbar?
	Explizite Datentypen	#potentieller Probleme mit Datentypen	Wie viele potentielle Probleme gibt es bei der Wahl der Datentypen und Festkommaskalierungen?
	Performance	Speicherverbrauch	Geht der generierte Code effizient mit dem Speicher um?
	Verwendung des Data Dictionaries	#Festkommaskalierungen mit/ohne DD-Eintrag	Wie viele Festkommaskalierungen sind (nicht) zentral im Targetlink-Data Dictionary hinterlegt?
		#Ports ohne DD-Eintrag	Wie viele Ports haben keinen zentral im Targetlink-Data Dictionary hinterlegten Datentyp?
Korrektheit	Einhaltung der Architektur	#Architekturverletzungen	Gibt es Unterschiede zur geplanten Architektur (z.B. einem SysML-Blockdiagramm)?
	Funktionale Korrektheit	#Anforderungen	Wie viele Anforderungen gibt es für das Modell (lässt Rückschlüsse auf die Problemgröße zu)?
		#Umgesetzter Anforderungen	Wie viele der Anforderungen sind im Modell umgesetzt?
Robustheit	Behandlung von Unter-/ Überläufen	#Sättigungsblöcke	Wie viele Sättigungsblöcke werden zur Behandlung von Über- und Unterläufen verwendet?
Testbarkeit	Ausreichende Tests	#Testfälle	Wie viele Testfälle gibt es für das Modell (ist interessant im Verhältnis zu #Anforderungen)?
		Testabdeckung	Wie gut wird das Modell getestet?
Verständlichkeit	Dokumentation	#Linienanmerkungen	Wie viele der Linien im Modell sind beschriftet?
		#Modellinformationsblöcke	In wie vielen Subsystemen wurden Modellinformationsblöcke zur Dokumentation verwendet?
		#Subsystemanmerkungen	In wie vielen Subsystemen wurden Anmerkungen (zur Erhöhung der Verständlichkeit) gemacht?
	Kognitive Erfassbarkeit	ØVernetzungsgrad der In- und Outports	Auf wie viele In-/Outports wirkt sich durchschnittlich ein In-/Outport aus?
		#Verwendeter Blocktypen	Wie viele der möglichen zur Verfügung stehenden Blocktypen wurden in dem Modell verwendet?
		#Rückkopplungen	Wie viele Rückkopplungen sind in dem Modell enthalten?
		Tiefe der Subsystemhierarchie	Wie tief ist die Subsystemhierarchie (ist interessant im Verhältnis zu #Anforderungen)?
	Lesbarkeit	ØBreite/Höhe der Subsystemdarstellung	Wie breit/hoch (in Pixel) sind die Subsysteme auf dem Monitor im Durchschnitt?
		#Überkreuzender Linien	Wie viele überkreuzende Linien enthält das Modell?
	Umfang	ØBreite der In-/Output-Schnittstelle	Wie viele In-/Outports haben die Subsysteme des Modells im Durchschnitt?
		ØSignale pro Bus	Wie viele Signale enthält ein Bus in dem Modell im Durchschnitt?
		#Blöcke	Wie viele Blöcke enthält das Modell (ist interessant im Verhältnis zu #Anforderungen)?
		#Buserstellungen	Wie oft wird ein neuer Bus erstellt?
		#interne Zustände	Wie viele interne Zustände hat das Modell?
		#Linien	Wie viele Linien gibt es im Modell?
		#Subsysteme	Wie viele Subsysteme enthält das Modell (ist interessant im Verhältnis zu #Anforderungen)?
		Zyklomatische Komplexität	Wie groß ist die zyklomatische Komplexität der einzelnen Subsysteme und des Gesamtsystems?
	Verwendung sprechender Bezeichner	ØNamenswechsel pro Signal	Wie oft werden Signale umbenannt?
		#magischer Konstanten	Wie viele magische Konstanten (d.h. Konstanten ohne Namen) werden verwendet?
		#Konstanten mit Name	Wie viele Konstanten mit Name werden verwendet?
Wartbarkeit	Standardkonformität	#Verwendeter Modellierungsmuster	Wie oft wurden etablierte Modellierungsmuster verwendet?
		#Verletzter Modellierungsrichtlinien	Wie oft werden die gewählten Modellierungsrichtlinien verletzt?
	Vermeidung impliziter Konstrukte	#From- und Goto-Blöcke	Wie viele From- und Goto-Blöcke werden verwendet, die den normalen Signalfluss unterbrechen?
		#Verwendetem eingebettetem C-Code	Wie oft wurde bestehender C-Code in das Modell integriert?
		#Verwendung von Speicherblöcken	Werden viele Speicherblöcke verwendet, die nicht Teil des normalen Signalflusses sind?
	Vermeidung von Redundanz	#(fast) identischer Subsysteme/Blockgruppen	Wurden große bzw. viele Modellteile per Copy&Paste erstellt?
		#Gebrochener Bibliotheksverknüpfungen	Wie oft wurden Bibliotheksblöcke verwendet, die nachträglich verändert wurden?
		#unverwendeter Signale auf tiefster Ebene	Wie viele Signale werden nur durch Subsysteme durchgeschleust und nicht verwendet?
		#Verwendeter Bibliotheken	Wie viele Bibliotheken wurden insgesamt verwendet?
		#Verwendeter Bibliotheksblöcke	Wie oft wurden Blöcke aus Bibliotheken verwendet?

enthält sowohl das Simulink-Modell als auch externe Datenquellen (sog. *Data-Provider*). Die Data-Provider stellen direkt Messwerte oder aufbereitete Informationen über Modelle zur Verfügung. Zwei Data-Provider sind z. B. die Simulink Verification and Validation Toolbox, welche die zyklomatische Komplexität von Modellen misst, und der Modellierungsrichtlinenprüfer MXAM von MES[1], welcher die Konformität von Modellen prüft.

Die Metriken erheben ihre Messwerte sowohl durch Berechnungen direkt auf den Modellen als auch durch Abfrage von Informationen von den Data-Providern. Die Struktur der Simulink-Modelle wird mit einem Metamodell beschrieben.

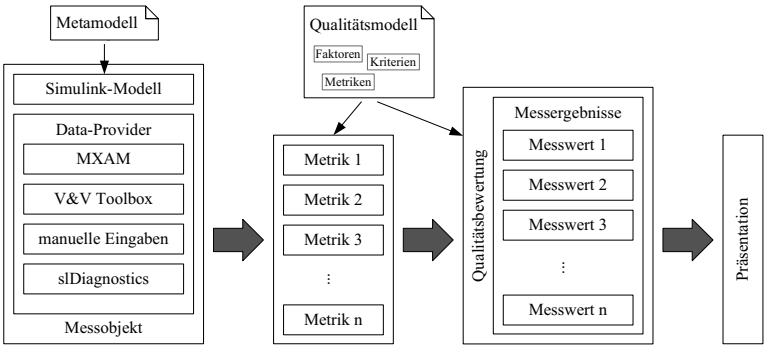

Abbildung 2: Schematischer Ablauf der Erhebung

Dieses Metamodell stellt in unserer Implementierung eine vereinfachte Sicht auf die Simulink-Modelle bereit und ermöglicht die Erhebung der Messwerte in einem Java-Prototypen. Als Grundlage dient der Simulink-Parser der TU-München aus dem ConQAT-Projekt[2]. Da der TUM-Parser weder das Nachladen von referenzierten Blöcken aus Bibliotheken unterstützt noch komfortable Zugriffsmethoden besitzt, wurde sein Metamodell in unserem eigenen Metamodell gekapselt. Dadurch sind die gewünschten Funktionen verfügbar und es ist z. B. möglich, den Wert eines Konstanten-Blocks auszulesen, unabhängig davon, ob er ein Simulink- oder Targetlink-Konstantenblock ist.

Mit Hilfe der Messergebnisse wird schließlich die Modellqualitätsbewertung durchgeführt. Dazu wird für jeden Messwert geprüft, ob er innerhalb der erlaubten Grenzen liegt. Dann wird für jeden Faktor berechnet, wie viele seiner Metriken Messwerte innerhalb ihrer erlaubten Grenzen haben. Das Ergebnis der Modellqualitätsbewertung wird anschließend in der grafischen Oberfläche des Prototypen präsentiert.

4 Auswertung der Messergebnisse

Abbildung 3 zeigt erste beispielhafte Messergebnisse, die mit dem Prototyp ermittelt wurden. Dabei ist zu beachten, dass aus Platzgründen nicht alle bereits umgesetzten Metriken aus

[1] http://www.model-engineers.com/en/our-products/model-examiner.html
[2] http://conqat.cs.tum.edu/index.php/SimulinkLibrary

	#potentieller Probleme mit Datentypen	#Targetlink-Funktionen	#Skalierungen mit DD-Eintrag	#Skalierungen ohne DD-Eintrag	#Ports ohne DD-Eintrag	#Anforderungen	#Testfälle	#Linienanmerkungen	#Subsystemanmerkungen	ØBreite/Höhe der Subsystemdarstellung	#Überkreuzender Linien	#Blöcke	#Buserstellungen	#interne Zustände	#Linien	#Subsysteme	#magischer Konstanten	#From- und Goto-Blöcke	#Verwendung von Speicherblöcken	#Gebrochener Bibliotheksverknüpfungen	#Verwendeter Bibliotheksblöcke
Modell 1	711	6	37	0	63	339	317	5.131	96	161	904	18.779	81	389	19.429	1.924	24	0	70	9.170	10.578
Modell 2	2.984	13	57	96	993	1.651	527	10.280	994	149	2.545	38.427	438	478	41.677	3.732	4	288	175	17.707	20.149
Modell 3	1.236	2	15	20	36	503	192	3.688	145	137	1.116	10.551	250	428	11.757	978	39	150	21	4.305	4.799

Abbildung 3: Messwerte von Simulink-Modellen aus einer sehr frühen Entwicklungsphase

Abbildung 1 enthalten sind und noch keine Modellqualitätsbewertung durchgeführt wurde.

Für eine robuste Modellqualitätsbewertung werden mehr Messwerte benötigt. Erst dann können die erlaubten Grenzen für die einzelnen Metriken festgelegt werden. Deshalb müssen mehr Messwerte gesammelt werden, um die Modellqualitätsbewertung zu validieren. Dabei können z. B. Reviewergebnisse, Experteneinschätzungen oder Fehleranzahlen aus Tests als Referenz verwendet werden. Wenn die Modellqualitätsbewertung nicht zutreffend ist, müssen die Grenzen der Metriken und/oder das Qualitätsmodell angepasst und dann erneut auf die Messwerte angewendet werden. Das Vorgehen ist iterativ so oft zu wiederholen, bis die Modellqualitätsbewertung mit den Referenzwerten übereinstimmt.

5 Verwandte Arbeiten

Ein weiteres generisches Qualitätsmodell stellt der Goal-Question-Metric-Ansatz (GQM) dar [BCR94]. Durch den Ansatz wird wie bei der Verwendung eines FCM-Modells die Qualität von Software bzw. Modellen objektiv erfassbar.

Zusätzlich zu dem Qualitätsmodell von Cavano und McCall gibt es viele weitere ähnliche Softwarequalitätsmodelle. Zwei Beispiele sind das Softwarequalitätsmodell von Boehm in [BBL76] und die darauf basierende ISO 9126. Allen gemeinsam sind Faktoren wie Zuverlässigkeit, Effizienz, Wartbarkeit und Portierbarkeit. Unterschiede gibt es jedoch bei den weiteren spezifischen Faktoren und dem Zusammenspiel der Kriterien.

Deißenböck et al. [DWP+07] stellen ein Qualitätsmodell für die Wartbarkeit auf. Es enthält nicht nur die Qualitätskriterien, sondern auch deren Einfluss auf die nötigen Wartungsaktivitäten. In einer Fallstudie erweitern sie ihr Qualitätsmodell um Simulink-spezifische Kriterien und Aktivitäten. Die automatisierte Qualitätsbewertung steht nicht in ihrem Fokus.

Einen Ansatz zur Qualitätsbewertung von UML-Modellen beschreiben Lange und Chaudron in [LC05]. Sie unterscheiden zwischen Entwicklung und Wartung, da in den jeweiligen Phasen andere Qualitätskriterien relevant sind. Sie verwenden zum großen Teil OO-Metriken und nur einige modellspezifische Metriken, wie z. B. die Anzahl der überkreuzenden Linien.

6 Zusammenfassung und Ausblick

Wir haben in dieser Arbeit ein Qualitätsmodell zur automatisierten Qualitätsbewertung von Simulink-Modellen vorgestellt. Das Qualitätsmodell enthält auf unterster Ebene Metriken, welche Messwerte auf den Simulink-Modellen messen. Es wurde der schematische Ablauf der Messung vorgestellt und kurz auf ein Metamodell für die Simulink-Modelle eingegangen. Schließlich wurden erste Messergebnisse vorgestellt.

Wie in Abschnitt 4 beschrieben, ist der nächste Schritt die Kalibrierung der einzelnen Metriken, d. h. das Ermitteln der erlaubten Grenzen. Außerdem muss der Zusammenhang zwischen der Problemgröße (z. B. Anzahl der Anforderungen) und den erlaubten Grenzen der Metriken weiter untersucht werden. Des Weiteren zeigen erste empirische Ergebnisse, dass die Ausdrucksmächtigkeit des Qualitätsmodells noch erhöht werden muss, um z. B. mit widersprüchlichen Messwerten besser umgehen zu können. Ebenso ist zu prüfen, ob verschiedene Entwicklungsphasen in dem Qualitätsmodell berücksichtigt werden müssen.

Literatur

[BBL76] B. W. Boehm, J. R. Brown und M. Lipow. Quantitative evaluation of software quality. In *Proceedings of the 2nd International Conference on Software Engineering*, Seiten 592–605, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[BCR94] V. Basili, G. Caldiera und H.D. Rombach. *Encyclopedia of Software Engineering*, Kapitel Measurement, Seiten 646–661. John Wiley & Sons, Inc., 1994.

[CM78] J. P. Cavano und J. A. McCall. A framework for the measurement of software quality. In *Proceedings of the Software Quality Assurance Workshop on Functional and Performance Issues*, Seiten 133–139, 1978.

[DWP+07] F. Deißenböck, S. Wagner, M. Pizka, S. Teuchert und J.-F. Girard. An Activity-Based Quality Model for Maintainability. In *Proc. of the 23rd IEEE International Conference on Software Maintenance*. IEEE CS Press, 2007.

[FHR08] F. Fieber, M. Huhn und B. Rumpe. Modellqualität als Indikator für Softwarequalität: eine Taxonomie. *Informatik-Spektrum*, 31(5):408–424, October 2008.

[FLS01] K. Frühauf, J. Ludewig und H. Sandmayr. *Software-Projektmanagement und –Qualitätssicherung*, Kapitel Software-Nutzen und -Kosten. Teubner Stuttgart, 2001.

[LC05] C. F. J. Lange und M. R. V. Chaudron. Managing Model Quality in UML-Based Software Development. In *Proc. of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, Seiten 7–16. IEEE Computer Society, 2005.

[Rau01] A. Rau. Einsatz von Metriken bei der modellbasierten Software-Entwicklung. *Simulationstechnik - 15. Symposium in Paderborn*, 2001.

[Sch10] J. Scheible. Ein Framework zur automatisierten Ermittlung der Modellqualität bei eingebetteten Systemen. In *Proc. of the Dagstuhl-Workshop: Model-Based Development of Embedded Systems (MBEES), 2010, Schloss Dagstuhl, Germany*, 2010.

[SVEH07] T. Stahl, M. Völter, S. Efftinge und A. Haase. *Modellgetriebene Softwareentwicklung: Techniken, Engineering, Management*. dpunkt.verlag, 2007.