# Komplexitätsmetriken für Geschäftsprozessmodelle*

Volker Gruhn, Ralf Laue

Universität Leipzig, Lehrstuhl für Angewandte Telematik und E-Business†
{gruhn,laue}@ebus.informatik.uni-leipzig.de

**Abstract:** Häufig werden grafische Modelle zum Software-Entwurf verwendet. In späten Phasen der Softwareerstellung sind solche Modelle oft schon recht "codenahe". Gerade in den früheren Phasen gibt es aber häufig Modelle, deren Hauptzweck darin besteht, die Kommunikation zwischen Auftraggebern und Entwicklern zu unterstützen. Um diese Kommunikation zu erleichtern, sollten die Modelle möglichst leicht verständlich sein. Hierfür ist zunächst zu klären, was man unter „leicht verständlich“ versteht. Unser Beitrag untersucht, inwiefern sich Ideen bekannter Software-Komplexitätsmetriken auf grafische Geschäftsprozessmodelle übertragen lassen.

## 1 Einführung

Ein Hauptzweck von Modellen in der Softwareentwicklung ist es, die Kommunikation zwischen den Prozessbeteiligten zu unterstützen. Hierfür sollten die Modelle möglichst leicht verständlich sein. Es ist daher wünschenswert, eine Metrik zu entwickeln, die eine Aussage über die Komplexität von Modellen liefert. Für Software sind zahlreiche Komplexitätsmetriken bekannt, die erfolgreich eingesetzt werden. Wir untersuchen, inwiefern sich die diesen Metriken zugrunde liegenden Ideen auf Geschäftsprozessmodelle (GPM) übertragen lassen. Wir geben einen Überblick über mögliche Komplexitätsmetriken und zeigen Richtungen für künftige Forschung.

## 2 Größe des Modells

Die einfachste Komplexitätsmetrik für Software, Lines of Code (LOC), bestimmt die Zahl der Codezeilen bzw. Anweisungen. Für GPM lohnt es, Metriken zu betrachten, die zu den LOC analog sind, also etwa die Zahl der im Modell abgebildeten Aktivitäten. Es ist naheliegend, dass Modelle mit weniger Aktivitäten leichter zu verstehen sind. Allerdings sagt die Zahl der Aktivitäten noch nichts über den Kontrollfluss zwischen den Aktivitäten aus. Da aber gerade alternative und parallele Abläufe das Verständnis erschweren können, untersuchen wir im Folgenden die Komplexität der Kontrollflussstruktur.

*Extended Abstract, vollständiger Artikel abrufbar auf `ebus.informatik.uni-leipzig.de/~laue`

†Der Lehrstuhl für Angewandte Telematik / E-Business ist ein Stiftungslehrstuhl der Deutschen Telekom AG

## 3 Kontrollflussstruktur des Modells

McCabe[McC76] schlug die zyklomatische Zahl als Software-Komplexitätsmetrik vor. Sie zählt die möglichen Wege, die im Kontrollflussgraphen zurückgelegt werden können und entspricht der Zahl der binären Entscheidungen (also der `if`-Statements) plus 1.

Cardoso[Car05] leitet hiervon eine analoge Metrik für GPM ab. Er betrachtet den Kontrollfluss zwischen den Aktivitäten. Im Allgemeinen gibt es in einem GPM Splits, die parallele oder alternative Kontrollflüsse starten, und Joins, die diese wieder zusammenführen. [Car05] berechnet eine Metrik, indem für jeden Split die Zahl der möglichen von diesem Split ausgehenden Kontrollflüsse ermittelt wird. Für einen AND-Split ist diese Zahl 1 (immer *alle* folgenden Pfade werden parallel bearbeitet ), für einen XOR-Split mit n Ausgängen n (da es n Möglichkeiten gibt, einen der Pfade auszuwählen) und für einen OR-Split mit n Ausgängen $2^n - 1$ (das entspricht den Möglichkeiten, mindestens einen und höchstens n Pfade auszuwählen). Die so ermittelten Zahlen werden dann addiert.

Die Zahl der Entscheidungsmöglichkeiten sagt aber noch immer eher wenig über die *Struktur* des Modells aus. Von dieser hängt aber die Verständlichkeit entscheidend ab.

## 4 Innere Struktur des Modells

Für Kontrollstrukturen von Software sind die Metriken „maximale und mittlere Verschachtelungstiefe“ bekannt. Eine Übertragung auf GPM ist leicht möglich: Die Verschachtelungstiefe beschreibt dann die Zahl der Kontrollflussentscheidungen, die getroffen werden müssen, damit eine Kontrollflusskante durchlaufen werden kann.

Weiterhin ist zu beachten, dass in manchen GPM Splits und Joins nicht paarweise auftreten, was das Verständnis erschwert. Bei einer Bestimmung der Komplexität eines GPM ist also zu berücksichtigen, ob Kontrollflussblöcke „sauber verschachtelt“ sind oder „unstrukturierte Sprünge“ auftreten.

[MRWH79] führte zur Messung der Häufigkeit von „unstrukturierten Sprüngen“ in Computerprogrammen die Knotenzähl-Metrik ein. Ein Kontrollflussgraph hat einen Knoten, wenn sich Pfade, entlang derer Kontrollfluss stattfinden kann, überschneiden.

Für GPM definiert [Aal98] den Begriff der *Wohlstrukturiertheit*. Diese setzt voraus, dass Splits und Joins paarweise „ordentlich verschachtelt“ vorkommen. Formell bedeutet dies, dass ein Petri-Netz, in das das GPM überführt wurde, frei von Handles [1] ist. Folglich liefert die Zahl der Handles eine Metrik für die Zahl der nicht wohlstrukturierten Konstrukte. Diese Metrik ist in ihrer Bedeutung vergleichbar mit der Knotenzähl-Metrik. Eine hohe Zahl von Handles deutet auf ein schlechtes Modell hin. Für wohlstrukturierte Modelle ist die Metrik „Zahl der Handles“ stets 0.

[1] In diesem Überblicksbeitrag verzichten wir bewusst auf die formale Definition. Kurz gesagt, tritt ein Handle in einem Petri-Netz dann auf, wenn es zwischen einer Stelle und einer Transition zwei verschiedene Pfade gibt, die außer dem Anfangs- und dem Endpunkt keine gemeinsamen Bestandteile haben. Details findet man in [ES89] und [Aal98]. Als Zahl der Handles in einem Petri-Netz definieren wir die Zahl der Paare aus einer Stelle und einer Transition mit der genannten Eigenschaft.

## 5 Erfassbarkeit des Modells

[SW03] schlägt das kognitive Gewicht als eine Software-Komplexitätsmetrik vor. [SW03] ordnet grundlegenden Software-Kontrollstrukturen ein kognitives Gewicht zu, das den geistigen Aufwand, dieses Konstrukt zu erfassen, misst. Das kognitive Gewicht einer Softwarekomponente wird definiert als Summe der kognitiven Gewichte ihrer Bestandteile. Es erscheint uns vielversprechend, diese Metrik auf GPM zu übertragen, allerdings fehlen in [SW03] noch einige Konstrukte, die für GPM von Bedeutung sind (etwa Cancellation).

In [PKG+00] wird untersucht, wie viele bekannte Designmuster in einem Modell vorkommen. Da die Verwendung von erprobten Designmustern die Verständlichkeit und Wartbarkeit einer Software erhöht, wird bei Verwendung solcher Muster eine positive Auswirkung auf die Komplexität angenommen. Allerdings erfordern solche Metriken eine umfangreiche Erfahrung mit diesen Mustern, um sinnvoll gedeutet werden zu können. Neben den „guten“ Mustern untersucht [PKG+00] auch das Gegenteil: als „schlecht“ bekannte Designmuster (Anti-Patterns), die auf mangelhafte Programmierung hindeuten.

Eine Übertragung der Gedanken aus [PKG+00] auf GPM halten wir für sinnvoll, wobei uns insbesondere das Entdecken der „schlechten“ Anti-Patterns hilfreich erscheint.

## 6 Modularisierung des Modells: Fan-in / Fan-out-Metriken

In graphischen GPM wird häufig der Gesamtprozess sinnvoll in kleinere Teilprozesse untergliedert. Diese Modularisierung kann helfen, den Gesamtprozess besser zu analysieren.

Um ein in Teilmodule gegliedertes Softwaresystem zu analysieren, bestimmt [HK81] für ein Software-Modul die Werte Fan-in und Fan-out. *Fan-in* ist die Zahl der fremden Module, die das zu bewertende Modul aufrufen. *Fan-out* ist die Zahl der Aufrufe anderer Module aus dem zu bewertenden Modul heraus. Weist nun ein Modul sowohl einen großen Fan-in- als auch einen großen Fan-out-Wert auf, ist das ein Indikator für schlechte Modularisierung. [HK81] nutzt die Maßzahl $((fan-in)\cdot(fan-out))^2$ als Metrik, die geeignet ist, solche Defekte aufzuspüren. Diese Metrik kann analog für GPM genutzt werden.

## 7 Zusammenfassung und weiterführende Fragestellungen

Tab. 1 stellt *Software*-Komplexitätsmetriken analoge Metriken für *GPM* gegenüber. Wie auch bei der Messung von Softwarekomplexität, gibt es auch für GPM nicht eine einzige universelle Metrik, die alle wesentlichen Einflussgrößen berücksichtigt. Es ist daher sinnvoll, die Metriken kombiniert anzuwenden. Da sich die Software-Komplexitätsmetriken in der Praxis bewährt haben, erwarten wir, dass die Anwendung analoger Ansätze für GPM ebenfalls zu guten Ergebnissen führt. Eine Validierung dieser Erwartung ist Gegenstand unserer weiterführenden Arbeiten.

Einige der vorgeschlagenen Metriken lassen sich sofort einsetzen. Beim kognitiven Ge-

| Software-Metrik | GPM-Metrik | Anwendung, Bewertung |
|---|---|---|
| Lines of Code | Zahl der Aktivitäten | sehr einfache Metrik, Kontrollfluss nur ungenügend berücksichtigt |
| zyklomatische Zahl [McC76] | Metrik nach Cardoso [Car05] | berücksichtigt Komplexität an Entscheidungsknoten, daher gut geeignet zur Bestimmung der Zahl der zum Testen nötigen Testfälle |
| Verschachtelungstiefe | Verschachtelungstiefe | gut geeignet als Ergänzung zur Metrik von Cardoso |
| Knotenzahl [MRWH79] | Zahl der Handles | misst „Wohlstrukturiertheit“ (Ein- und Aussprünge in bzw. aus Kontrollstrukturen) |
| kognitives Gewicht [SW03] | verallg. kognitives Gewicht | misst den von einem Leser eines Modells zu erfassenden „Informationsgehalt“ des Modells |
| (Anti)Patterns [PKG+00] | (Anti)Patterns für GPM | erfordert Erfahrung; Anti-Patterns geeignet, um mangelhafte Modellierung aufzudecken |
| Fan-in / Fan-out [HK81] | Fan-in / Fan-out | geeignet, um schlechte Unterteilung des Modells in Teilmodelle aufzudecken |

Tabelle 1: Komplexitätsmetriken für Software und Geschäftsprozessmodelle

wicht sowie der Bewertung von Patterns und Anti-Patterns sind Anpassungen an GPM-Belange nötig. Diese sind Gegenstand unserer weiterführenden Arbeiten.

## Literatur


[Aal98] Wil M.P. van der Aalst. The Application of Petri Nets to Workflow Management. *The Journal of Circuits, Systems and Computers*, 8(1):21–66, 1998.

[Car05] Jorge Cardoso. How to Measure the Control-flow Complexity of Web Processes and Workflows. In *The Workflow Handbook*, Seiten 199–212, 2005.

[ES89] Javier Esparza und Manuel Silva. Circuits, handles, bridges and nets. In *Applications and Theory of Petri Nets*, Seiten 210–242, 1989.

[HK81] S Henry und K Kafura. Software Structure Metrics based on Information Flow. *IEEE Transactions on Software Engineering*, 7(5):510–518, 1981.

[McC76] Thomas J. McCabe. A Complexity Measure. *IEEE Trans. Software Eng.*, 2(4):308–320, 1976.

[MRWH79] M. A. Hennell M. R. Woodward und D. Hedley. A measure of control-flow complexity in program text. *IEEE Transactions on Software Engineering*, SE-5(1):45–50, 1979.

[PKG+00] Jukka Paakki, Anssi Karhinen, Juha Gustafsson, Lilli Nenonen und A. Inkeri Verkamo. Software Metrics by Architectural Pattern Mining. In *Proc. International Conference on Software: Theory and Practice*, Seiten 325–332, 2000.

[SW03] Jingqiu Shao und Yingxu Wang. A New Measure of Software Complexity based on Cognitive Weights. *IEEE Canadian Journal of Electrical and Computer Engineering*, 28(2):69–74, 2003.