Objektorientierte Codemetriken und deren Anwendung im Bereich von Embedded Systems

Eugenia Drosdezki, 166771
Angewandte Informatik, WS2008/2009
Seminar Neue Technologien, Hochschule Offenburg

16. Januar 2009

In der Softwareentwicklung können Prozess- und Produkt-Metriken dabei helfen, sowohl Verwaltungstätigkeiten wie Kalkulation, Personalmanagament und Controlling als auch Entwicklungstätigkeiten wie Analyse, Entwurf, Implementierung, Dokumentation und Test zu bewerten[AM96].

Empirische Untersuchungen haben gezeigt, dass Metriken als Indikatoren der Softwarequalität verwendet werden können. Durch die immer zunehmende Popularität objektorientierter Paradigmen wird gerade die Bewertung der Qualität objektorientierter Software immer wichtiger [HCN98].
Objektorientierte Metriken berücksichtigen die strukturelle Besonderheit und Charakteristika objektorientierter Software und helfen dabei potenzielle Schwachstellen in der Software zu identifizieren und den Softwareentwurf zu optimieren. Der Einsatz objektorientierter Metriken senkt die Entwicklungs-, Test- und Wartungskosten [CK94].

In dieser Arbeit werden objektorientierte Metriken von Chidamber und Kemerer, Li und Henry sowie von Bruito und Abreu näher vorgestellt. Danach werden die Ergebnisse einer Studie erläutert, in der gezeigt wird, dass Metriken-Tools, die unter anderem Metriken von Chidamber und Kemerer messen, Definitionen von Metriken unterschiedlich interpretieren. Zum Schluss wird noch aufgezeigt, wie objektorientierte Codemetriken im Bereich eingebetteter Systeme sinnvoll eingesetzt werden können.

1 Einleitung

In der Softwareentwicklung werden Methoden und Techniken erforscht und eingesetzt, die die Softwarequalität erhöhen und die Kosten und den Projektverlauf besser kalkulierbar machen sollen. Eines der wichtigsten Ziele in der Softwareentwicklung ist die Kontrolle des Entwicklungsprozesses, damit auch der Kosten, Aufgaben und der Qualität [LH93].

Um Softwarequalität beurteilen zu können, wurden Qualitätsmerkmale definiert (Tabelle 1). Die Kriterien Funktionalität, Laufzeit, Zuverlässigkeit und Benutzbarkeit bilden die Qualitätssicht des Anwenders, da sie unmittelbar nach außen sichtbar sind. Die Kriterien Transparenz, Übertragbarkeit, Wartbarkeit und Testbarkeit spiegeln die inneren Werte des Softwareproduktes wieder. Sie sind zwar nach außen hin nicht unmittelbar erkennbar, sind aber maßgeblich für den langfristigen Erfolg eines Softwareproduktes verantwortlich. Einige Kriterien schließen sich gegenseitig aus, andere korrelieren miteinander. Die Software-Qualitätssicherung kennt bereits viele Methoden und Techniken zur Erkennung von heute vorherrschenden Softwarefehlern und teilt sich in die Bereiche Produktqualität und Prozessqualität auf. Ein Bereich zur Sicherung der Produktqualität ist die statische Analyse. Bei statischen Analyseverfahren werden syntaktische oder semantische Programmeigenschaften aus dem Quelltext abgeleitet.

kundenorientiert		**herstellerorientiert**
Funktionalität		Übertragbarkeit
Zuverlässigkeit		Änderbarkeit
Effizienz		Testbarkeit
Benutzbarkeit		Transparenz

Tabelle 1: Qualitätsmerkmale eines Softwareprodukts [Hof08]

Der Programmcode wird nicht ausgeführt. Ein Teilgebiet der statischen Analyse sind die Codemetriken. Sie ermöglichen es, die oben genannten Qualitätsmerkmale quantitativ zu erfassen. Dadurch werden unsichtbare Aspekte der Software sichtbar und auf einer objektiven Ebene vergleichbar [Hof08].

Objektorientierte (OO) Metriken gehören zu den Codemetriken [Hof08]. Durch die stetig zunehmende Popularität objektorientierter Paradigmen wird die Bewertung der Qualität objektorientierter Software immer wichtiger [HCN98]. OO Metriken berücksichtigen die strukturelle Besonderheit und Charakteristika objektorientierter Software und helfen dabei potenzielle Schwachstellen in der Software zu identifizieren und den Softwareentwurf zu optimieren. Diese Metriken können in unterschiedliche Kategorien aufgeteilt werden. Es gibt unter anderem Umfangs-, Vererbungs-, Kopplungs- und Kohäsionsmetriken.

Inhalt: Abschnitt 2 stellt die wichtigsten Veröffentlichungen vor. In Abschnitt 3 wird die Metriken Suite von Chidamber und Kemerer vorgestellt. Abschnitt 4 beschäftigt sich mit den Metriken vorgestellt von Li und Henry. Abschnitt 5 präsentiert die MOOD Metriken-Suite von Brito und Abreu. In Abschnitt 6 werden einige Metriken-Tools aufgezeigt, die die Metriken von Chidamber und Kemerer berechnen. In Abschnitt 7 wird der Einsatz von Metriken im Embedded Bereich behandelt. Abschnitt 8 beinhaltet die Zusammenfassung.

2 Metriken

Die heute noch verwendete Metrik Lines-of-Code (LOC) war die erste Codemetrik. 1955 wurde diese Metrik zur Analyse des ersten FORTRAN Compilers eingesetzt. Die MCCabe Cyclomatic Number und die Halstead-Metrik sind weitere bekannte Metriken. Sie wurden 1976 und 1977 eingeführt [Zus98]. Metriken aus dieser Zeit wurden für die prozedurale Programmierung entwickelt und lassen sich in der objektorientierten Sprache nur auf der Ebene der Methodenimplementierungen einsetzen. Sie können ein System als ganzes nicht erfassen, da die Aspekte des objektorientierten Entwurfs nicht gemessen werden. Hierzu werden Metriken benötigt, die die spezifischen Eigenschaften objektorientierter Programmiersprachen abbilden [Hof08].

Metriken zur Messung objektorientierter Strukturen be-

rücksichtigen objektorientierte Konzepte wie Klassen, Methoden, Vererbung, Polymorphismus, Überladung und Kapselung. Ende der achtziger Jahre wurden die ersten Metriken zur Messung objektorientierter Strukturen vorgestellt. Rocacher veröffentlichte bereits 1988 eine Studie zu objektorientierten Metriken. 1989 definierte Morris 9 OO Metriken[Zus98]. Chidamber und Kemerer entwickelten 6 Metriken zum Messen des objektorientierten Designs und evaluierten diese analytisch nach Grundsätzen von Weykers. Sie publizierten die erste Version dieser Metriken 1991 und veröffentlichten 1994 nochmals eine Arbeit mit überarbeiteten Definitionen für einen Teil der Metriken [CK94]. Sharble und Cowen veröffentlichten 1993 eine Publikation über Metriken für objektorientiertes Design. Das erste Buch über objektorientierte Metriken wurde 1994 von Lorenz vorgestellt. Danach folgte Henderson-Sellers 1996 mit seinem Buch.

Heute existieren zahlreiche OO Metriken. Das von Horst Zuse entwickelte System ZD_MIs stellt z.B. Informationen von 200 objektorientierten Metriken bereit[Zus98]. Da nicht alle existierenden Metriken aufgezählt werden können, werden hier nur einige Sammlungen genauer behandelt. Es wird die Metriken-Suite von Chidamber und Kemerer, Metriken von Li und Henry, die MOOD Suite von Brito und Abreu näher vorgestellt.

Bei der Auswahl geeigneter Metriken ist zu beachten, dass Metriken erst erfolgreich eingesetzt werden können, wenn sie empirisch validiert wurden. Ohne eine solche Grundlage ist eine sinnvolle Analyse mit Metriken nicht möglich [HCN98]. Viele der publizierten objektorientierten Metriken sind nicht ausreichend validiert.

3 Metriken von Chidamber und Kemerer [CK94]

Die wohl bekannteste und am meisten zitierte Metriken-Sammlung ist die von Chidamber und Kemmerer (CK). In Anlehnung an Wand und Weber wählten Chidamber und Kemmerer die ontologischen Prinzipien von Bunge als theoretische Grundlage und entwickelten sechs Design-Metriken. Anschließend evaluierten sie die vorgeschlagenen Metriken mit Hilfe von empirischen Daten analytisch gegen Wykers vorgeschlagenen Satz von Mess-Prinzipien. Die CK Metriken wurde bereits in vielen empirischen Studien validiert [SK03]. Churcher und Shepperd kritisierten die CK Metriken. Deren Ansicht nach sind diese unpräzise und mehrdeutig definiert. Sie wünschen sich, dass klare Leitlinien für die Anwendung der Metriken für bestimmte Programmiersprachen bereitgestellt werden [CS95].

Kategorie	Metrik	Beschreibung
Umfang	WMC	Weighted Methods per Class
	RFC	Response of a Class
Vererbung	DIT	Dept of Inheritance Tree
	NOC	Number of Childeren
Kopplung	CBO	Coupling between Object Classes
Kohäsion	LCOM	Lack of Cohesion in Methods

Tabelle 2: CK Metriken

3.1 WMC - Weighted Methods per Class

Die gewichtete Methodenkomplexität erfasst die Summe der Komplexitäten der Methoden einer Klasse. Für eine Klasse C mit den Methoden $M_1, M_2, M_3, ..., M_n$ ist $C_1, C_2, C_3, ...C_n$ die Komplexität der Methoden. Die Methode zur Berechnung der einzelnen Komplexitäten wurde in der Arbeit von Chidamber and Kemerer nicht vorgegeben.

$$WMC = \sum_{i=1}^{n} c_i \quad (1)$$

Bewertung Die Anzahl und Komplexität der Methoden ist ein Indikator für den Aufwand, der zur Entwicklung und Wartung einer Klasse notwendig ist. Klassen mit vielen Methoden weisen eine erhöhte Komplexität auf. Sie haben einen hohen Einfluss auf ihre Kindklassen.

3.2 DIT - Dept of Inheritance Tree

Wenn man die Vererbungshierarchie im objektorientierten Design als Baum modelliert, sind die Knoten dieses Baumes Klassen. Für jede Klasse ist DIT die Länge des Pfades vom Knoten bis zur Wurzel der Hierarchie. Im Fall der Mehrfachvererbung wird der längste Pfad gewählt. DIT misst, wie viele Vorfahren eine Klasse potenziell beeinflussen. Für die Wurzel des Baumes ist DIT 0.

Bewertung Je tiefer sich eine Klasse in der Vererbungshierarchie befindet, desto mehr Methoden werden geerbt. Dies macht es schwer, das Verhalten einer Klasse zu verstehen. Ein Baum mit vielen Klassen unten in der Vererbungshierarchie ist fehleranfälliger und schwierig zu testen, da sich Änderungen am Vorfahren auch auf die Kinder auswirken können. Hat ein Baum viele Elemente in Wurzelnähe, also einen niedrigen DIT-Wert, ist er zwar leichter zu verstehen, nutzt aber den Vorteil der Wiederverwendung durch Vererbung nicht.

3.3 NOC - Number of Childeren

Diese Metrik steht für die Anzahl direkter Nachfahren einer Klasse.

Bewertung Die Anzahl der Kinder zeigt den potenziellen Einfluss einer Klasse an. Je mehr Subklassen eine Klasse besitzt, desto umfangreicher sollte sie getestet werden. Da Fehler in dieser Klasse sich auch auf die Kind-Klassen auswirken können. Andererseits nutzen Klassen mit wenig Kindelementen den Vorteil der Wiederverwendung durch Vererbung nicht.

3.4 CBO - Coupling between Object Classes

Mit dieser Metrik wird die Anzahl der Klassen, die an eine Klasse gekoppelt sind, berechnet. Eine Kopplung entsteht, wenn eine Klasse Methoden oder Instanzvariablen einer anderen Klasse aufruft.

Bewertung Übermäßige Kopplung verhindert die Wiederverwendung von Code. Je unabhängiger eine Klasse ist, desto eher kann man sie in einer anderen Anwendung wiederverwenden. Um das modulare Design nicht zu beeinträchtigen und zur Förderung der Kapselung, sollte die Kopplung minimal gehalten werden. Eine große Anzahl gekoppelter Klassen erhöht die Empfindlichkeit auf Veränderungen in anderen Bereichen. Solcher Code ist schwer zu warten. Module mit einer hohen Kopplung sollten intensiver getestet werden.

3.5 RFC - Response of a Class

Diese Metrik bezeichnet die Anzahl aller Methoden, die von einer Klasse potenziell ausgeführt werden können, wenn ein Objekt dieser Klasse auf eingegangene Nachrichten reagiert. Sie wird berechnet aus der Anzahl aller lokalen Methoden mit der Anzahl der Methoden, die von den lokalen Methoden aufgerufen werden.

Bewertung Eine höhere Kennzahl steht für eine höhere Komplexität der Klasse und macht das Testen und Debuggen komplizierter, da eine längere Zeit notwendig ist, um die Klasse zu verstehen.

3.6 LCOM - Lack of Cohesion in Methods

Der innere Zusammenhalt einer Klasse ist charakterisiert durch die Anzahl lokaler Methoden, die auf Instanzvariablen zugreifen. Sie wird berechnet aus der Anzahl der Paare von lokalen Methoden einer Klasse mit gemeinsamen Instanzvariablen subtrahiert von der Anzahl der Paare von lokalen Methoden einer Klasse ohne gemeinsame Instanzvariablen. Wenn die Subtraktion ein negatives Ergebnis ergibt, wird der Wert auf 0 gesetzt.

Für die Klasse C_1 mit n Methoden $M_1, M_2, M_3, ..., M_n$ ist $\{I_i\}$ die Menge aller Instanzvariablen, die von der Methode M_i verwendet werden. Es gibt n solcher Mengen $\{I_1\}$, $\{I_2\}$, ..., $\{I_n\}$.
Sei P = { (I_i,I_j) | $I_i \cap I_j = \emptyset$ } und Q = { (I_i,I_j) | $I_i \cap I_j \neq \emptyset$}. Wenn alle Mengen $\{I_1\}$, $\{I_2\}$, ..., $\{I_n\}$ $\emptyset$ sind dann ist P = $\emptyset$. $1 \leq$ i, j $\leq$ n

$$LCOM = \begin{cases} |P| - |Q|, \text{ wenn } |\mathrm{P}| > |\mathrm{Q}| \\ 0, \text{ wenn } |\mathrm{P}| \leq |\mathrm{Q}|. \end{cases} \quad (2)$$

Bewertung Ein hoher Wert ist wünschenswert, da dadurch die Kapselung gefördert wird. Geringe Kohäsion erhöht die Komplexität. Es wird empfohlen die Klasse in mehrere Klassen zu unterteilen.

4 Metriken von Li und Henry [LH93]

Li und Henry verwendeten die Metriken aus der Sammlung von Chidamber und Kemerer (außer CBO) [CK94] und fünf weitere Metriken. Sie untersuchten, ob objektorientierte Metriken dazu geeignet sind, den Wartungsaufwand zu bestimmen. Für die Gewichtung von WMC wurde McCabes Cyclomatic Number verwendet. Auf der Basis einer empirischen Untersuchung bestätigten sie, dass objektorientierte Metriken, insbesondere die CK Metriken, für die Einschätzung des Wartungsaufwands nützlich sind [FP97].

Kategorie	Metrik	Beschreibung
Umfang	NOM	Number of Local Methodes
	Size1	Number of Semicolons in a Class
	Size2	Number of Attrubutes + NOM
Kopplung	MPC	Message Passing Coupling
	DAC	Data Abstraction Coupling

Tabelle 3: Metriken von Li und Henry

4.1 MPC - Message Passing Coupling

Wenn ein Objekt einer Klasse Methoden eines anderen Objektes aufruft, wird zwischen den beiden Objekten eine Nachricht verschickt. Diese Metrik berechnet die Anzahl der Aufrufe zum Verschicken von Nachrichten in einer Klasse. Die Anzahl von einer Klasse gesendeter Nachrichten kann den Grad der Abhängigkeit der lokalen Methoden von Methoden anderer Klassen anzeigen.

4.2 DAC - Data Abstraction Coupling

Diese Metrik misst die Anzahl der Attribute in einer Klasse, die abstrakte Datentypen (ADT) sind. Die Abstraktion von Daten ist eine Technik, die es ermöglicht, neue Datentypen wie z.B. Klassen zu erzeugen.

$$DAC = Anzahl\ Abstrakter\ Datentypen\ definiert\ in\ einer\ Klasse \quad (3)$$

DAC zeigt die Anzahl der Datenstrukturen an, die von Definitionen anderer Klassen abhängig sind. Je mehr abstrakte Datentypen in einer Klasse definiert sind, desto größer ist die Kopplung dieser Klasse mit anderen Klassen.

4.3 NOM - Number of local Methodes

In dieser Metrik wird die Anzahl der Instanzmethoden in einer Klasse berechnet. Sie ermöglicht Schlussfolgerungen über die Komplexität einer Klasse.

4.4 SIZE1 - Number of Semicolons in a Class

Diese Metrik ist eigentlich die traditionelle Lines of Code (LOC) Metrik. Sie wird durch die Anzahl der Semikolons in einer Klasse berechnet.

4.5 SIZE2 - Number of Attriubutes + Number of Local Methods

Diese Metrik ist ein Maß für die Anzahl der Eigenschaften einer Klasse. Man berechnet sie aus der Summe der Attributen und Methoden einer Klasse.

5 MOOD Metriken von Brito und Abreu [AM96][HCN98]

Die von Brito und Abreu eingeführten MOOD Metriken (Metrics for Object Oriented Design) arbeiten auf Systemebene. Sie können gerade für Projektleiter zur Gesamtbewertung eines Systems von Nutzen sein. Das Metriken-Set bezieht sich auf grundlegende strukturelle Mechanismen des objektorientierten Paradigmas wie Kapselung (MHF und AHF), Vererbung (MIF und AIF), Polymorphismus (POF) und Message-Passing (COF).

Abreu und Melo bewiesen mit Hilfe der MOOD Metriken in einer Studie den Einfluss objektorientierten Designs auf die Softwarequalität. Anhand der aus dem Experiment gewonnenen Daten zeigten sie, dass Entwurfstechniken wie Vererbung, Polymorphismus, Kapselung und Kopplung Qualitätsmerkmale wie die Zuverlässigkeit und Wartbarkeit beeinflussen können. Sie suchten nach Korrelationen mit den Qualitätsattributen Fehlerdichte und die benötigte Zeit zur Fehlerbehebung.

Kategorie	Metrik	Beschreibung
Vererbung	MIF	Method Inheritance Factor
	AIF	Attribute Inheritance Factor
Kopplung	COF	Coupling Factor
Kapselung	MHF	Method Hiding Factor
	AHF	Attribute Hiding Factor
Polymorphie	POF	Polymorphism Factor

Tabelle 4: Objektorientierte MOOD Metriken

5.1 MHF - Methode Hiding Factor

Mit dieser Metrik wird das Verhältnis der verborgenen Methoden zur Anzahl aller Methoden angegeben. MHF und AHF wurden als Maß der Kapselung vorgeschlagen und sollten gemeinsam ausgewertet werden. Der MHF-Zähler ist die Summe

der verborgenen Methoden aus allen Klassen. Die Verborgenheit einer Methode ist der Prozentsatz der gesamten Klassen für die diese Methode nicht sichtbar ist. Der MHF-Nenner ist die Anzahl der Methoden im ganzen System.

$$MHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{M_d(C_i)} (1 - V(M_{mi}))}{\sum_{i=1}^{TC} M_d(C_i)} \tag{4}$$

$$V(M_{mi}) = \frac{\sum_{j=1}^{TC} is\_visible(M_{mi}, C_j)}{TC - 1} \tag{5}$$

$$is\_visible(M_{mi}, C_j) = \begin{cases} 1, \text{ wenn j} \neq \text{i} \wedge C_j \text{ darf } M_{mi} \text{ aufrufen} \\ 0, \text{ sonst.} \end{cases} \tag{6}$$

TC = Anzahl Klassen
$M_d(C_i)$ = Anzahl nicht geerbter Methoden in einer Klasse
$V(M_{mi})$ = % der Klassen für die die Methode M_{mi} sichtbar ist (in C++ 1 für public Methoden und 0 für private Methoden).

Bewertung Messungen haben gezeigt, dass MHF negativ mit der Fehlerdichte korreliert. Bei der Erhöhung des Wertes verringert sich die Fehlerdichte und die Zeit, die benötigt wird um Fehler zu beheben.

5.2 AHF - Attribute Hiding Factor

Diese Metrik gibt das Verhältnis der verborgenen (private oder protected) Attribute zur Anzahl aller Attribute wieder.

$$AHF = \frac{\sum_{i=1}^{TC} \sum_{m=1}^{A_d(C_i)} (1 - V(A_{mi}))}{\sum_{i=1}^{TC} A_d(C_i)} \tag{7}$$

$$V(A_{mi}) = \frac{\sum_{j=1}^{TC} is_v isible(A_{mi}, C_j)}{TC - 1} \tag{8}$$

$$is\_visible(A_{mi}, C_j) = \begin{cases} 1, \text{ wenn j} \neq \text{i} \wedge C_j \text{ darf } A_{mi} \text{ aufrufen} \\ 0, \text{ sonst} \end{cases} \tag{9}$$

$A_d(C_i)$ = Anzahl nicht geerbter Attribute in einer Klasse
$V(A_{mi})$ = % der Klassen für die das Attribut A_{mi} sichtbar ist.

Bewertung Bei AHF wurden bei den Messungen von [AM96] keine nennenswerten Korrelationen gefunden. Allerdings kann man durch Kapselung komplexer Komponenten Seiteneffekte reduzieren.

5.3 MIF - Methode Inheritance Factor

Mit dieser Metrik wird das Verhältnis der geerbten Methoden zur Anzahl aller Methoden angegeben. Diese Metrik ist ein Maß für die Wiederverwendung in einem System. Im Rahmen der Validierung von Kitchenham zeigte sich, dass MIF und AIF als direktes Maß für die Vererbung gültig sind.

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} (M_d(C_i) + M_i(C_i))} \tag{10}$$

TC = Anzahl Klassen
$M_i(C_i)$ = Anzahl geerbter und nicht überschriebener Methoden in einer Klasse
$M_d(C_i)$ = Anzahl deklarierter Methoden in einer Klasse

Bewertung MIF korreliert negativ mit der Ausfalldichte und hat eine hohe Korrelation mit der Fehlerdichte und Nacharbeit. Das bedeutet, dass bei der Erhöhung dieses Wertes sich die Fehlerdichte und die notwendige Zeit zum Debuggen verringern sollten. Allerdings sollte die Vererbung sparsam eingesetzt werden. In [AM96] wird vermutet, dass ein zu hoher MIF-Wert (über 70-80%) die positive Wirkung umkehrt.

5.4 AIF - Attribute Inheritance Factor

Diese Metrik gibt das Verhältnis der geerbten Attribute zur Anzahl aller Attribute wieder.

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} (A_d(C_i) + A_i(C_i))} \tag{11}$$

TC = Anzahl Klassen
$A_i(C_i)$ = Anzahl geerbter und nicht überschriebener Attribute in einer Klasse
$A_d(C_i)$ = Anzahl deklarierter Attribute in einer Klasse

Bewertung Die Messungen in [AM96] ergaben für AIF eine geringe negative Korrelation mit der Ausfalldichte und eine moderate negative Korrelation mit der Fehlerdichte und Nacharbeit. Für dieses Ergebnis konnten keine ausdrücklichen Schlussfolgerungen gezogen werden.

5.5 COF - Coupling Factor

COF misst die Kopplung zwischen Klassen in einem System, wobei die Kopplung aufgrund von Vererbung nicht eingerechnet wird. Diese Metrik gibt das Verhältnis vorhandener Verbindungen zwischen Klassen zur Anzahl aller potenzieller Verbindungen im System wieder. Zur Berechnung von COF werden alle möglichen Paare von Klassen gebildet und auf Message Passing oder auf semantische Verbindungen (eine Klasse referenziert auf Attribute oder Methoden einer anderen Klasse) überprüft. COF ist also ein Maß für die Kopplung zwischen zwei Klassen für alle paarweisen Beziehungen zwischen den Klassen in einem System und wurde wie folgt definiert:

$$COF = \frac{\sum_{i=1}^{TC} \left[\sum_{j=1}^{TC} is_c lient(C_i, C_j) \right]}{TC^2 - TC} \tag{12}$$

mit

$$is\_client(C_c, C_s) = \begin{cases} 1, \text{ wenn } C_c \Rightarrow C_s \wedge C_c \neq C_s \\ 0, \text{ sonst} \end{cases} \tag{13}$$

$C_c \Rightarrow C_s$ steht für die Verbundung zwischen einer Client-Klasse C_c und einer Lieferanten-Klasse C_s.

Bewertung COF hat eine hohe positive Korrelation mit der Nacharbeit, Ausfall- und Fehlerdichte. Daher bewirkt eine erhöhte Kopplung zwischen den Klassen einen Anstieg in der Fehlerdichte und Nacharbeit. Dieses Ergebnis zeigt, dass Kopplung in Softwaresystemen einen starken negativen Einfluss auf die Softwarequalität hat und daher auf ein erforderliches Minimum reduziert werden sollte. Viele gekoppelte Beziehungen zwischen Klassen erhöhen die Komplexität und verringern die Kapselung und potenzielle Wiederverwendung.

5.6 POF - Polymorphism Factor

POF wird definiert durch die Anzahl überschriebener geerbter Methoden dividiert durch die maximale Anzahl aller möglichen disjunkten polymorphen Möglichkeiten. POF ist indirekt ein Maß dynamischer Bindungen in einem System. Es ist für Systeme ohne Vererbung undefiniert.

$$POF = \frac{\sum_{i=1}^{TC} M_o(C_i)}{\sum_{i=1}^{TC} [M_n(C_i) \times DC(C_i)]} \quad (14)$$

$DC(C_i)$ = Anzahl der Nachkommen
$M_n(C_i)$ =neue Methoden
$M_o(C_i))$ =überschriebene Methoden

Bewertung POF hat eine mäßige bis hohe Korrelation mit der Nacharbeit, Fehler- und Ausfalldichte. Das bedeutet, dass eine angemessene Nutzung des Polymorphismus beim objektorientierten Entwurf die Anzahl der Fehler und die notwendige Nacharbeit verringern sollte. Von hohen Werten von POF (über 10%) wird eine Verringerung der Vorteile erwartet.

6 Metriken-Tools [LLL08]

In der Studie von Lincke, Lundberg und Löwe wurden einige kommerzielle und frei verfügbare Metriken-Tools nach bestimmten Kriterien ausgewählt und auf drei unterschiedlich große Software-Systeme angewendet. Die berechneten Werte zeigten, dass für die gleichen Software-Systeme und Metriken von Tool zu Tool unterschiedliche Messwerte generiert werden. Bei der Auswahl geeigneter Werkzeuge für ihre Studie haben Lincke, Lundberg und Löwe durch eine Internetrecherche 46 Werkzeuge gesammelt. Etwa die Hälfte dieser Werkzeuge berechnet nur einfache Werte wie Lines-of-Code. Die andere Hälfte berechnet zusätzlich anspruchsvollere Metriken (OO Metriken). Sie haben festgestellt, dass die meisten Werkzeuge Messungen für Java-Programme durchführen können, weitere analysieren C/C++, UML oder andere Programmiersprachen. Aus der Auswahl von 46 Werkzeugen wurden die Tools ausgewählt, die Java (Source- oder Bytecode) unterstützen, bekannte objektorientierte Metriken auf Klassenebene berechnen und frei verfügbar sind oder eine Evaluationslizenz anbieten. Da einige Werkzeuge nur wenige Dateien gleichzeitig verarbeiten können, wurde die Auswahl nochmal auf 10 Werkzeuge eingegrenzt.

Die Analyse dieser Metriken-Tools ergab, dass Definitionen von objektorientierten Metriken unterschiedlich interpretiert und implementiert werden. Bei der Verwendung dieser Werkzeuge muss man beachten, dass die berechneten Werte werkzeugabhängig sind und dadurch entsprechend unterschiedliche Ergebnisse in der Beratung liefern. Um dies zu vermeiden, müssten alle Metriken-Tools die vorgeschlagenen Metriken auf die gleiche Weise implementieren, wie sie validiert wurden.

Werkzeuge	**Metriken**						
Name	CBO	DIT	NOC	LCOM	NOM	RFC	WMC
Analyst4j	x	x	x	x	x	x	x
CCCC	x	x	x		x		
Chidamber & Kemmerers Java Metrics	x	x	x	x	x	x	
Dependency Finder		x	x		x		
Eclipse Metrics Plugin 1.3.6		x	x		x		x
OOMeter	x	x	x	x			
VizzAnalyzer	x	x	x	x	x	x	x

Tabelle 5: Ein Teil der in [LLL08] verwendeten Werkzeuge und Metriken

Analyst4j: Das Tool basiert auf der Eclipse Plattform als stand-alone Rich Client Anwendung oder als Eclipse IDE plug-in. Es verfügt über eine Suche, Metriken, eine Qualitätsanalyse für Java Programme und erzeugt Reporte.

CCCC: CCCC ist ein Open Source Kommandozeilen-Tool. Es analysiert C++ und Java Dateien und generiert Reporte.

Chidamber & Kemmerers Java Metrics: Dieses Tool ist Open Source und wird über die Kommandozeile bedient. Es berechnet die Metriken von Chidamber und Kemmerer während der Verarbeitung des Java-Bytecodes.

Dependency Finder: Dieses Tool ist Open Source. Es ist eine Suite von Werkzeugen zum Analysieren von kompilierten Java Code.

Eclipse Metrics Plugin 1.3.6: Das Open Source Eclipse IDE plug-in berechnet Metriken und analysiert Abhängigkeiten.

OOMeter: Dieses Tool ist ein experimentelles Software-Metriken Werkzeug. Es akzeptiert UML Modelle in XMI, Java und C# Quellcode und berechnet verschiedene Metriken.

VizzAnalyzer: Der VizzAnalyzer ist ein Analysetool für Qualität. Es liest Softwarecode und andere Design Spezifikationen sowie Dokumentationen und führt eine Reihe von Qualitätsanalysen aus.

7 Anwendung Objektorientierter Codemetriken in Embedded-Software

Heutzutage beinhalten bereits viele Produkte eingebettete Systeme. Der Markt für solche Produkte wird voraussichtlich in den nächsten 10 Jahren exponentiell anwachsen. Durch die immer größer werdende Leistungsfähigkeit der verwendeten Microcontroller wächst auch die Größe der Programme. Die Menge und Vielfalt der Software in eingebetteten Systemen steigt. Die Funktionalität eingebetteter Systeme wird zunehmend in Software umgesetzt. Bereits seit Jahren übersteigt der Aufwand für die Entwicklung von Softwarekomponenten den Aufwand für die Entwicklung von Hardwarekomponenten erheblich[Hof08] [GLT03] [ORea08]. Im Automobilbau ist der Anteil softwaregesteuerter Innovationen in den Jahren 1999 bis 2003 von 20% auf 80% gestiegen [SKS03].

In den kommenden Jahren ist die Fähigkeit zur erfolgreichen Entwicklung eingebetteter Systeme mit hoher Qualität in kurzer Zeit der Schlüssel zum Erfolg. Allerdings haben immer mehr Unternehmen Probleme Produkte mit guter Qualität pünktlich zu liefern. Zur Optimierung der Aktualität, Produktivität und Qualität eingebetteter Software Entwicklung müssen Unternehmen geeignete Softwareentwicklungstechnologien einsetzen [GLT03]. Um die neuen Anforderungen an die Qualitätssicherung zu erfüllen, wurde z.B. bei Volkswagen ein Vorgehensmodell zur Softwarequalitätssicherung entwickelt, dass unter anderem Metriken zur Qualitätsverbesserung verwendet. Das Vorgehensmodell wurde mit einem Tool realisiert, dass mit Hilfe von Qualitätsmetriken und Programmierregeln potenziell risikobehaftete Module und Strukturen im Sourcecode erkennt [SKS03].

In den meisten Bereichen der Softwareentwicklung konnten objektorientierte Sprachen und Techniken bereits erfolgreich genutzt werden. Im Bereich eingebetteter Systeme haben Entwickler Bedenken objektorientierte Techniken einzusetzen, da

sie enorme Performanceeinbußen und zusätzlichen Code befürchten. Das Paper von Murray und Shahabuddin [MS06] zeigt den erfolgreichen Einsatz objektorientierter Techniken und Methoden zur Entwicklung einer eingebetteten Software für einen Satelliten. Sie haben festgestellt, dass der Einsatz objektorientierter Entwicklungsmethoden unter anderem die Kommunikation mit System- und Hardwareentwicklern erleichterte. Außerdem ermöglichte die Verwendung einer objektorientierten Sprache, Code auf einem einfachen Weg wiederzuverwenden und Plattforminformationen innerhalb einer Klasse von plattformunabhängigen Schnittstellen zu kapseln. Der objektorientierte Entwurf förderte die Aufteilung des Systems in unabhängige Komponenten mit sichtbaren, gut dokumentierten Schnittstellen. Diese Eigenschaft wiederum vereinfachte die Automatisierung auf Unit-, Modul- und System-Ebene. Um den Ressourcenverbrauch klein zu halten, wurde die Software in Embedded C++ implementiert. Gegenüber C++ bietet diese Programmiersprache eingeschränkten Funktionsumfang [MS06].

Dass der Einsatz von Metriken zur Verbesserung der Softwarequalität beiträgt, wurde in vorherigen Kapiteln bereits erläutert. Im Bereich von Embedded Systems werden vorwiegend Metriken zum Messen von physikalischen Eigenschaften wie Performance, Speicher- und Energieverbrauch eingesetzt. Weitere Metriken, die erfolgreich eingesetzt werden können sind Wiederverwendung, Time-To Market und Kosten. Vorgeschlagene Qualitätsmetriken konnten bereits erfolgreich zur Verbesserung der Softwarequalität verwendet werden. Allerdings ist zu beachten, dass gerade die Software-Qualitätsmerkmale Wiederverwendung, Abstraktion, Kopplung und Kohärenz negative Auswirkungen auf die Performance, auf den Speicherverbrauch und auf andere Physikalische Metriken haben können. Die von Oliveira, Redin, Carro und einigen anderen Autoren vorgestellte Arbeit beschäftigt sich damit, einen Vergleich zwischen objektorientierten Metriken und physikalischen Eigenschaften eingebetteter Systeme aufzustellen. Ziel dieser Studie ist es, Korrelationen der objektorientierten Metriken mit den physikalischen Metriken zu analysieren, um Entwürfe eingebetteter Systeme bewerten zu können. Es wurde festgestellt, dass gerade die Design-Phase eingebetteter Systeme von der Nutzung objektorientierter Metriken profitiert. Diese Metriken können dazu beitragen, den Wiederverwendungsfaktor zu erhöhen, die Entwicklungskosten zu reduzieren und die Wartbarkeit zu verbessern. Die Verwendung objektorientierter und physikalischer Metriken hilft den Entwicklern ein besseres Verständnis für das richtige Gleichgewicht zwischen Wiederverwendung und Optimierung zu finden. Allerdings ist die Verwendung OO Metriken im Embedded Bereich noch nicht weit verbreitet [ORea08].

In einer weiteren Studie von Redin, Oliveira, Brisolara und anderen wird gezeigt, wie Qualitätsmetriken bei den Entscheidungen verwendet werden können, physikalische Eigenschaften eines eingebetteten Systems zu verbessern [ROea08].

Zhang und Jarzabek beschrieben die Ergebnisse eines Experiments, in dem vier verschiedene mobile Geräte, die Rollenspiel-Anwendungen ausgeführen, im Hinblick auf die Performance und Softwarequalität analysiert wurden. Zur Bewertung der Softwarequalität wurden objektorientierte Metriken verwendet und aus den gewonnenen Ergebnissen Verbesserungen am Softwaredesign durchgeführt. Die Studie zeigt, dass durch Wiederverwendung von Plattform- und oder Softwarekomponenten der Entwicklungsaufwand ohne Kompromisse in der Performance stark reduziert werden kann [ZJ05].

8 Zusammenfassung

Empirische Studien haben gezeigt, dass der Einsatz objektorientierter Metriken sich positiv auf Softwarequalitätsmerkmale wie Wartbarkeit, Testbarkeit und Wiederverwendbarkeit objektorientierter Software-Systeme auswirkt. Sie helfen den Entwicklern potenzielle Schwachstellen der Software zu identifizieren und das System besser zu verstehen. Design-Metriken können verwendet werden um den Entwurf objektorientierter Software zu analysieren und an die gegebenen Anforderungen anzupassen. Gerade bei der Entwicklung eingebetteter Software-Systeme hilft die Bewertung des Entwurfs in der Design-Phase, um das richtige Gleichgewicht zwischen Softwarequalität und Systemoptimierung zu finden. Es hat sich herausgestellt, dass objektorientierte Metriken gerade im Bereich eingebetteter Systeme wenig Anklang finden, da die Verwendung objektorientierter Programmiersprachen im Vergleich zu prozeduralen Programmiersprachen mehr Code erzeugt und Performance-Einbußen mit sich bringt. Die zunehmende Komplexität eingebetteter Systeme und der immer höher werdende Softwareanteil zwingt die Entwickler dazu, neue Techniken für die Softwareentwicklung um zur Verbesserung ihrer Qualität einzubeziehen.

In dieser Arbeit wurden einige Sammlungen von Metriken vorgestellt. Tabelle 6 zeigt eine Übersicht über alle hier erwähnten Metriken. Metriken können erst erfolgreich eingesetzt werden, wenn sie empirisch validiert wurden. Der Vergleich einiger Metriken-Tools hat gezeigt, dass die Ergebnisse einer Software-Analyse mit Hilfe solcher Werkzeuge stark von den jeweiligen Tools abhängen. Dies ergibt sich durch die unterschiedliche Interpretation und Implementierung der Definitionen objektorientierter Metriken. Um dies zu vermeiden müssten alle Metriken-Werkzeuge die vorgeschlagenen Metriken auf die gleiche Weise implementieren, wie sie validiert wurden. Wenn man bedenkt, dass die Ergebnisse der Metriken stark von der Implementierung der Werkzeuge abhängen, erweist sich die Validierung von Softwaremetriken als sehr schwierig. Um die Ergebnisse vergleichbar und verallgemeinbar zu machen, müsste neben den Metriken zusätzlich auch der Messprozess spezifiziert werden [LLL08]. Der Prozess der empirischen Validierung von objektorientierten Metriken dauert bis heute an.

Kategorie	**Metrik**	**Beschreibung**
Umfang	WMC	Weighted Methods per Class
	RFC	Response of a Class
	NOM	Number of Local Methodes
	Size1	Number of Semicolons in a Class
	Size2	Number of Attrubutes + NOM
Vererbung	DIT	Dept of Inheritance Tree
	NOC	Number of Childeren
	MIF	Method Inheritance Factor
	AIF	Attribute Inheritance Factor
Kopplung	CBO	Coupling between Object Classes
	COF	Coupling Factor
	MPC	Message Passing Coupling
	DAC	Data Abstraction Coupling
Kohäsion	LCOM	Lack of Cohesion in Methods
Kapselung	MHF	Method Hiding Factor
	AHF	Attribute Hiding Factor
Polymorphie	POF	Polymorphism Factor

Tabelle 6: Objektorientierte Metriken

Literatur

[AM96] Abreu, Fernando Brito und WalcClio Melo: *Evaluating the Impact of Object-Oriented Design on Software Quality.* In Proceedings of the 3 rd International Software Metrics Symposium, 1996.

[CK94] Chidamber, Shyam R. und Chris F. Kemerer: *A Metrics Suite for Object Oriented Design.* IEEE Trans. Software Engineering, 20(6), Juni 1994.

[CS95] Churcher, Neville I. und Martin J. Shepperd: *Comments on A Metrics Suite for Object Oriented Design.* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 21(3), März 1995.

[FP97] Fenton, Norman E. und Shari Lawrence Pfleeger: *Software Metrics.* PWS Publishing Company, 1997.

[GLT03] Graaf, Bas, Marco Lormans und Hans Toetenel: *Embedded Software Engineering: The State of the Practice.* IEEE Software, 20(6), November 2003.

[HCN98] Harrison, Rachel, Steve J. Counsell und Reuben V. Nithi: *An Evaluation of the MOOD Set of Object-Oriented Software Metrics.* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 24(6), Juni 1998.

[Hof08] Hoffmann, Dirk W.: *Software Qualität.* Springer, November 2008.

[LH93] Li, Wei und Sallie Henry: *Object Oriented Metrics which Predict Maintainability.* Department of Computer Science, Verginia Polytechnic Institute and State University, Februar 1993.

[LLL08] Lincke, Rüdiger, Jonas Lundberg und Welf Löwe: *Comparing software metrics tools.* In: *ISSTA*, Seiten 131–142. ACM, August 2008.

[MS06] Murray, Alexander T. und Mohammad Shahabuddin: *OO Techniques Applied to a Real-time, Embedded, Spaceborne Application.* California Institute of Technology, 2006.

[ORea08] Oliveira, Marcio F. S., Ricardo Miotto Redin und Luigi Carro et al.: *Software quality metrics and their impact on embedded software.* MOMPES, 2008.

[ROea08] Redin, Ricardo M., Marcio F. S. Oliveira und Lisane B. Brisolara et al.: *On the use of software quality metrics to improve physical properties of embedded systems.* Springer, 2008.

[SK03] Subramanyam, Ramanath und M.S. Krishnan: *Empirical Analysis of CK Metrics for Object-Oriented Design Complexity.* IEEE TRANSACTIONS ON SOFTWARE ENGINEERING, 29(4), April 2003.

[SKS03] Stücka, Renate, Jörg Kessler und Frank Schröder: *Achillesferse Bordelektronik-Softwarequalitätssicherung im Automobilbau.* Carl Hanser Verlag, München, 48, 2003.

[ZJ05] Zhang, Weishan und Stan Jarzabek: *Reuse without Compromising Performance: Industrial Experience from RPG SoftwareProduct Line for Mobile Devices.* LNCS 3714, 2005.

[Zus98] Zuse, Horst: *A Framework of Software Measurement.* Walter de Gruyter, 1998.