Seminar „Qualitative und Quantitative Methoden im Software Engineering“

Ausarbeitung zum Vortrag „Methoden und Metriken für Objektorientierte Systeme“ von Oliver Feuser.

Inhalt

- Zusammenfassung Abstract
- Konventionelle Metriken
  - Lines Of Code
  - McCabe's Cyclomatic Complexity
  - Fazit
- Function-Point Metrik
- Objektorientierte Maße
  - Weighted Methods per Class
  - Depth of Inheritance Tree
  - Number Of Children
  - Coupling Between Objects
  - Response For a Class
  - Lack of Cohesion in Methods
  - Metrikenkataloge
    - MOOSE
    - Sharble / Cohen
    - Li / Henry
- Grundlagen und Validation
- Vergleichbarkeit der Metriken
- Sprachspezifische Konstrukte
- Objektorientierte Metriken für Entwurf und Dokumentation
- Literatur

Zusammenfassung

Abstract

Um die im objektorientierten Bereich Verwendung findenden Strukturelemente adäquat bei der Software-Messung berücksichtigen zu können, reicht es nicht, die aus der prozeduralen Entwicklung bekannten Metriken auf die Klassen oder die zugehörigen Methoden eines Systems anzuwenden. Vielmehr ist es notwendig, neue, an die Strukturen der Objektorientierung angepaßte Metriken zu entwerfen, um auch die Beziehungen der Objekte (Systeme, Klassen, Methoden) untereinander zu berücksichtigen. Dennoch werden zur Messung einzelner Methoden die konventionellen Metriken benötigt, da die Methoden - als kleinste funktionale Einheit mit definiertem Eintritts- und Austrittspunkt - keine OO-Strukturen aufweisen.
Als Folge daraus entsteht ein Schnitt zwischen konventionellen und OO-Metriken: Für die Erfassung der Komplexität eines Systems müssen Metriken verwendet werden, die für die jeweilige Programmiertechnik ausgelegt sind. Dies bringt natürlich auch Probleme mit sich, da die Messergebnisse von objektorientierten Eigenschaften nicht mit den herkömmlichen Resultaten verglichen werden können.
Eine Ausnahme stellt die in der Software-Messung allgemein bekannte Function-Point Metrik dar. Sie ist unabhängig von einschlägigen Programmiertechniken und ermöglicht damit den direkten Vergleich verschiedener Entwicklungen. Überdies ermöglicht es diese Metrik, die Vorteile der objektorientierten Entwicklung (beispielsweise geringere Fehleranfälligkeit oder höhere Wiederverwendbarkeit, etc.) zu belegen, denn diese Annahmen sind noch lange nicht validiert.
Aber auch in Bezug auf die objektorientierten Metriken ist die experimentelle Validierung erst in einem frühen Stadium. Erst nach und nach stellt sich heraus, welche Maße geeignet sind und welche keine vernünftigen Aussagen über das Projekt erlauben. In gleicher Weise ist die theoretische Fundierung der objektorientierten Maße erst in einem Anfangsstadium. Wie sich dabei herausgestellt hat, stellen die sogenannten OO-Metriken keine Metriken im mathematischen Sinne dar, obwohl sie in der Literatur oft so bezeichnet werden. Aus diesem Grund werden auch in dieser Ausarbeitung die Begriffe Maß und Metrik synonym benutzt, obwohl dies aus mathematischer Sicht falsch ist.

Konventionelle Metriken

Als Vertreter der „konventionellen“ (d.h. nicht dedizierten OO-) Metriken werden in dieser Ausarbeitung die Metriken LOC (lines of code) und MCC (McCabe's cyclomatic complexity) behandelt.
Selbstverständlich lassen sich auch andere konventionelle Metriken auf OO-Entwicklungen anwenden. Da sich diese in der OO-Messung jedoch nicht grundlegend von den hier vorgestellten Metriken unterscheiden, wird auf eine weitergehende Vorstellung verzichtet.

LOC

Die älteste und bekannteste Metrik, LOC, läßt sich grundsätzlich in jeder bekannten Form für OO-Systeme verwenden ([LI93]), jedoch muß dabei auf die Erfassung von OO-Strukturen verzichtet werden.

Bei der Bewertung mit LOC muß unterschieden werden, ob das Maß auf Methodenebene bzw. auf Klassen- und Systemebene angewendet wird. Auf Methodenebene kann LOC ohne Einschränkung eingesetzt werden, da die Methoden in sich geschlossene Einheiten sind, die untereinander nur mit einer „Benutzt"-Relation verknüpft sein können. In der System- und Klassenebene hat die Objektorientierung jedoch massive Auswirkungen auf das Maß LOC. Einerseits ist es mit dieser Metrik nicht möglich, Strukturelemente, wie beispielsweise die Vererbung, auch nur annähernd in der Messung zu berücksichtigen. Andererseits kann LOC einfach auf das System angewendet werden, indem zum Beispiel die Anzahl der Statements in dem System gezählt werden oder indem die LOC-Meßwerte der zu den Klassen gehörenden Methoden akkumuliert werden. Anhand dieser zwei sehr unterschiedlichen Definitionen zur Messung einer Klasse oder eines Systems wird eine weitere Problematik von LOC für die Objektorientierung deutlich, da es nicht unerheblich ist, ob nur die Methoden oder der gesamte Programmcode der Klasse berücksichtigt wird, oder ob - als weitere Möglichkeit - das Maß LOC auf dem resultierenden Kontrollfuß in der jeweiligen Ebene angewendet wird. Es sei nur auf virtuelle Basisklassen verwiesen, die - ohne „echten" Programmcode ausgestattet - je nach Interpretation des Maßes mitgezählt werden können oder nicht.
Dieses Dilemma wird auch in der entsprechenden Literatur diskutiert ([MOR89], [TEG92]). Es gibt im wesentlichen zwei Standpunkte: die eine Seite, die LOC (und damit auch MCC sowie vergleichbare Metriken) für unbrauchbar hält, weil objektorientierte Strukturelemente nicht berücksichtigt werden, und die andere, die LOC - zu Recht - als gebräuchliche und validierte Metrik ansieht und sie aus diesen Gründen ohne Einschränkung auf OO-Entwicklungen anwendet, auch wenn dabei stark verfälschte Ergebnisse zu erwarten sind.

MCC

Die von McCabe entwickelte „cyclomatic complexity" (MCC) ([MCC76]) ist auf dem Kontrollflußgraphen eines Programms bzw. eines Moduls definiert. Als Maß wird die Zyklomatische Zahl des Graphen verwendet, die mit folgender Formel aus der Anzahl der Kanten e, der Anzahl der Knoten n und der Anzahl der unabhängigen Teilgraphen p berechnet wird zu
$MCC(M) = e_M - n_M + 2p_M$.
Analog zu LOC ist auch hier ersichtlich, daß auf die OO-Strukturelemente nicht eingegangen wird, der Unterschied liegt allerdings darin, daß MCC den Flußgraphen der betreffenden Ebene auswertet und somit hinsichtlich deren Komplexität eine prägnantere Aussage als LOC ermöglicht. In Bezug auf Klassen und Systeme gilt natürlich das entsprechend zu LOC gesagte, auch hier versagt die Metrik, wenn es um die Erfassung von beispielsweise Vererbungshierarchien geht. Auch wenn von der Erfassung dieser Strukturen abgesehen wird, bleibt zu bedenken, daß die Anwendung von MCC auf OO-Systeme allgemein einen höheren Meßwert als bei vergleichbaren konventionellen Systemen erzeugt. Dies gründet in dem meist komplexeren Kontrollfluß objektorientierter Software, der durch stärkere Abstraktion bedingt ist.
Eine ausführlichere Betrachtung der Übertragung von MCC auf den OO-Ansatz findet sich in [TEG92] und [ABR94].

Fazit

Wie anhand der Beispiele LOC und MCC vorgeführt, lassen sich die konventionellen Metriken nur sinnvoll auf die Methoden eines OO-Systems anwenden, da die verbleibenden Strukturmerkmale nicht oder nicht angemessen erfaßt werden können.
Da ein OO-System allerdings nicht nur als Anhäufung von Methoden unterschiedlicher Klassen zu sehen ist und die Maße auf Systemen und Klassen verfälschte Meßwerte liefern, sollte im allgemeinen von der Anwendung dieser Metriken außerhalb der Untersuchung von Methoden abgesehen werden, um Fehleinschätzungen zu vermeiden.

Function-Point Metrik

Die Function-Point Metrik ([IFP94], [DRE89]) unterscheidet sich von den bereits vorgestellten Metriken dadurch, daß der fertige Entwurf bzw. die Software gemessen wird, um den Entwicklungsaufwand, der für das angesetzte Projekt betrieben werden muß (beispielsweise Mann/Tage oder die Kosten des Projekts) im voraus abschätzen bzw. im nachhinein überprüfen zu können. Da sich die Function-Point Metrik aber auch dadurch auszeichnet, daß sie nicht nur von der verwendeten Programmiersprache, sondern auch von der Technologie, mittels derer ein Projekt entwickelt wird, unabhängig ist, ist die Function-Point Metrik ohne Einschränkung auf die Objektorientierung übertragbar. Ganz so, als wäre die Objektorientierung nicht benutzt worden. Trotzdem kann die Objektorientierung (je nach Zählweise und Projekt) das Ergebnis der Messung beeinflussen.
Gezählt werden bei dieser Metrik fünf extern (d.h. für den Anwender) sichtbare Aspekte der Software:

1. Die Eingaben (x4)
2. Die Ausgaben (x5)

3. Die Abfragen durch den Anwender (x4)
4. Die Datendateien, die durch die Anwendung verändert werden (x10)
5. Die Schnittstellen zu anderen Applikationen (x7)

Zusätzlich werden diese Gesichtspunkte mit einer Gewichtung versehen, um die unterschiedlichen Schwierigkeitsgrade bei der Implementierung annäherungsweise zu berücksichtigen (in Klammern angegeben).
Im einfachsten Fall der Software Entwicklung wird eine Applikation von Null an als einzelnes Projekt entwickelt. Bei einer derartigen Vorgehensweise ergibt die Applikations- und Projekt-Zählung das gleiche Ergebnis. Wird zum Beispiel ein System mit 1000 Function-Points entwickelt, so hat auch das Projekt 1000 Function-Points. Diese einfache Annäherung an die Applikations-Entwicklung wird durch die Objektorientierung mehr und mehr verdrängt.
Dies begründet sich in der Verfügbarkeit von Klassenbibliotheken, die dem Entwickler die Einbindung von vorgefertigten Objekten wie Dialogen, den zugehörigen Controls und der gleichen erheblich erleichtern. Die Objektorientierung ermöglicht darüber hinaus aber auch die (verhältnismäßig) einfache Einbindung einer Fülle von neuen Funktionalitäten, als Beispiel sei hier nur die Integration von OLE genannt.
In einem solchen Fall ist es schwierig, die Function-Points zu zählen, da nicht feststeht, ob die so eingebundene Funktionalität mitgezählt werden soll oder nicht. Eine Sichtweise besteht darin, einfach die so erzeugte Programm-Funktionalität in die Anwendungs- und Projekt-Zählung mit einzubeziehen. Dies ist durchaus sinnvoll bei Objekten, die vom Anwender ohnehin nicht als eigenständige Funktion erkannt werden, zum Beispiel Edit-Controls oder Buttons. Natürlich wird dies zu einem Problem, wenn umfangreiche Komponenten (beispielsweise komplette Dialoge oder ein Druck-Subsystem) eingebunden werden, von den Funktionen der heute üblichen Code-Grundgerüste vieler Entwicklungssysteme ganz zu schweigen. Problematisch bei diesem Verfahren ist, daß der Eindruck erweckt wird, daß der Aufwand für das Projekt größer ist als dies tatsächlich der Fall war. In der Literatur finden sich aus diesem Grund Stimmen, die von der Anwendung dieses Maßes abraten. Sie begründen dies damit, daß letztendlich der Anwender getäuscht wird, weil dieser das Projekt bzw. die Software anhand der Metrik beurteilt oder auch bezahlt.
Diese Problematik existiert natürlich schon länger als die Objektorientierung. Handelte es sich früher jedoch weitgehend nur um Ausnahmen, so werden die vorgestellten Möglichkeiten und Verfahren speziell in der Objektorientierung heute allgemein angewendet.

OO-Metriken

Die speziell für objektorientierte Systeme entwickelten Maße zeichnen sich dadurch aus, daß sie die Strukturmerkmale der Objektorientierung nicht nur beachten, sondern gezielt auswerten:

- Lokalisation: Das Plazieren von Dingen in gegenseitige physikalische Nähe.
  Der objektorientierte Ansatz plaziert Informationen um Objekte.
- Kapselung: Das Zusammenführen in eine Sammlung von Dingen.
  Objektorientierte Sprachen kapseln
  a. den Status
  b. angebotene Fähigkeiten (Methoden)
  c. andere Objekte
  d. Ausnahmen
  e. Konstanten
  f. Konzepte!
- Information Hiding: Das Verstecken von Details.
  Nur die Information wird zur Verfügung gestellt, die zur Erfüllung des Zieles notwendig ist.
  Kapselung und Information Hiding sind nicht das selbe!
- Vererbung: Ein Mechanismus, bei dem ein Objekt Eigenschaften eines anderen Objekt nutzt.
  Zu unterscheiden ist Einfach- und Mehrfachvererbung. Viele OO-Metriken basieren auf Vererbung, zum Beispiel:
  a. Anzahl der Kinder (direkte Spezialisierung)
  b. Anzahl der Eltern (direkte Generalisierung)
  c. Tiefe einer Klasse in der Vererbungshierarchie
- Abstraktion: Die Fokussierung auf die wichtigen Details eines Konzeptes oder eines Dinges.
  Abstraktion ist eine relatives Konzept; je stärker die Abstraktion, desto mehr werden Details ignoriert, d.h. desto genereller wird die Sichtweise. Mit abnehmender Abstraktion werden mehr Details eingeführt und damit Konzepte und Dinge spezifischer betrachtet.

Es wird offensichtlich, daß OO-Metriken in erster Linie auf Klassen bzw. Objekten operieren. Wich-

tiges Ziel der Maße ist es damit, die Klassen und Objekte eines Systems zu identifizieren, um sie bewerten zu können. In gleicher Weise müssen auch die Beziehungen der Klassen und Objekte untereinander identifiziert und geeignet bewertet werden. Damit lassen sich die OO-Metriken in Bezugsebenen aufteilen, indem

a. Maße auf Methodenebene
b. Maße auf Klassenebene
c. Maße auf Vererbungshierarchien
d. Maße auf Aggregationshierarchien

unterschieden werden, allerdings schließen sich die einzelnen Ebenen nicht gegenseitig aus. Die Maße auf Methodenebene beschränken sich auf die Messung von Eigenschaften einzelner Methoden. Hierzu werden in erster Linie die konventionellen Metriken, LOC und MCC, gezählt. Die Maße auf Klassenebene erweitern dies entsprechend auf die Strukturmerkmale einer Klasse und die Maße auf Vererbungshierarchien betrachten die durch die Vererbung (und Abstraktion) entstandenen Strukturen. Die Maße auf Aggregationshierarchien betrachten die Verknüpfungen der Klassen untereinander (nicht zu verwechseln mit der Vererbung).
Da es auch in diesem Bereich eine Vielzahl von unterschiedlichen Metriken gibt, kann auch hier wiederum nur eine Auswahl, wenn auch umfangreicher, vorgestellt werden. Im folgenden wird auf sechs ausgewählte Maße eingegangen. Die genauen Definitionen und weitere Betrachtungen zu diesen Maßen finden sich in [CHI94].

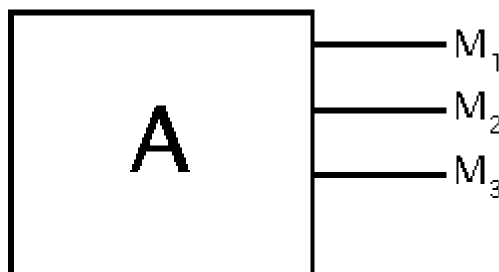
WMC

"Weighted Methods per Class"

Definition

WMC© = „Die Anzahl der Methoden in der Klasse *C* bzw. die Summe der jeweiligen Komplexität der einzelnen Methoden der Klasse *C*."
Die Berechnung der Komplexität einer Methode wird von den Autoren offen gelassen, dementsprechend gibt es für die Gewichtung unterschiedliche Vorschläge.
Die einfachste Methodenkomplexität stellt die konstante Komplexität 1 dar, es werden also nur die Methoden der Klasse durchgezählt, bzw. es wird angenommen, daß alle Methoden eine ähnliche Komplexität besitzen. WMC mit der Methodengewichtung 1 ist auch als NOM, „number of methods", ([LI93]) bekannt.
Gleichfalls wird McCabe's cyclomatic complexity als Methodenmaß vorgeschlagen, so daß die interne Struktur der Methoden berücksichtigt wird.

Beispiel



Offensichtlich besitzt die Klasse *A* dieser Abbildung drei Methoden. Damit gilt:
WMC(A) = g(M_1) + g(M_2) + g(M_3)
sofern *g* die Gewichtung der einzelnen Methoden angibt.

Eigenschaften des Maßes

- Anzahl und Komplexität der Methoden bestimmen den Aufwand für die Entwicklung und Wartung.
- Je größer die Anzahl der Methoden, desto größer ist der Einfluss auf Kinder, die die Methoden erben.
- Klassen mit vielen Methoden sind spezifischer als Klassen mit wenigen Methoden.
- Die Fehlerwahrscheinlichkeit nimmt mit der Anzahl der Methoden zu.

Kommentar

WMC stellt in der vorgestellten Form ein Maß auf Klassenebene dar, lässt sich aber auch auf die Aggregations- und die Vererbungshierarchie ausdehnen. Damit stellt es ein universell einsetzbares Maß dar, welches eine wesentliche Eigenschaft objektorientierter Entwicklung mißt und an die Phasen eines Projektes angepaßt werden kann.
Die Einschränkung auf eine konstante Gewichtung ist für die Phasen OOA (objektorientierte Analyse) und OOD (objektorientiertes Design) zu empfehlen, da diese Methodenkomplexität eine Unabhängigkeit von der Implementierung gewährleistet. Gleichfalls sollte für die Messung fertiger Programmfragmente ein dynamisches Methodenmaß eingesetzt werden, um die unterschiedliche Komplexität einzelner Methoden in das Meßergebnis mit einbeziehen zu können.

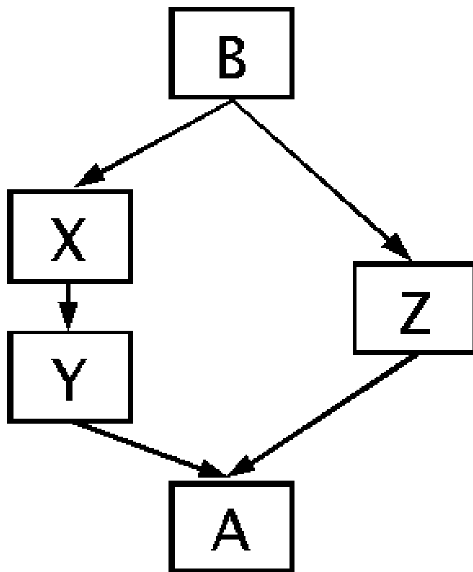
DIT

"Depth of Inheritance Tree"

Definition

DIT© = „Länge des maximalen Weges in der Klassenhierarchie von der Wurzel bis zur Klasse *C*."
Dabei wird die Tiefe der Klasse *C* in der Klassenhierarchie von der Wurzel aus mit 0 beginnend gezählt. Handelt es sich um eine Hierarchie mit Mehrfachvererbung, so ist der maximale Weg zwischen der Wurzel und der Klasse *C* zu zählen.

Beispiel



Im Beispiel dieser Abbildung gilt für die Basisklasse *B*:
Dit(*B*) = 0
Entsprechend sind
DIT(*X*) = DIT(*Z*) = 1
Für die Klasse *A* gilt
DIT(*A*) = 3
da der maximale Weg von *B* zu *A* über *X* und *Y* läuft.
Die vollständige Hierarchie *H* hat folglich eine Gesamttiefe von
DIT(*H*) = 3

Eigenschaften des Maßes

- Da die Basisklasse auch Eigenschaften für abgeleitete Klassen definiert, muß die Vererbungshierarchie bis zur Wurzel durchsucht werden. Nur so können alle Eigenschaften einer Klasse erfaßt werden.
- Der Gültigkeitsbereich von Attributen und Methoden ist abhängig von der Tiefe der Klassenhierarchie. DIT ist ein Maß für die Anzahl der Vorgänger, die Einfluß auf eine Klasse nehmen können.

Kommentar

Durch die Verwendung von Klassenbibliotheken und von Programmiersprachen, in denen Klassen stets als Spezialisierung von Systemklassen definiert werden müssen (zum Beispiel Smalltalk), ergeben sich höhere Meßwerte für die Klassen.
Als Anhaltspunkt für DIT gibt Lorenz einen Wert von 6 (sechs) an ([LOR94]). Auch wenn er empfiehlt, erst unterhalb der Bibliotheksklasse mit der Zählung zu beginnen, so muß der Wert einer derartigen starren Daumenregel zumindest hinterfragt werden. Sollen beispielsweise umfangreiche Systeme auf geometrischen oder mathematischen Objekten mit starker Abstraktion entworfen und implementiert werden, so ist im allgemeinen mit einer tieferen Klassenhierarchie zu rechnen.
Trotzdem handelt es sich bei DIT um ein Maß, welches mit Hinsicht auf die OO-Strukturmerkmale eine sinnvolle Aussage zu Wartbarkeit und Testaufwand ermöglicht.

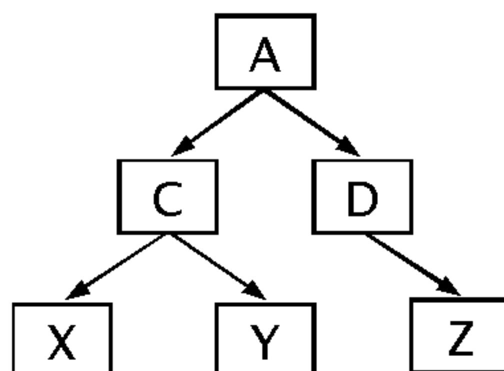
NOC

"Number Of Children"

Definition

NOC© = „Anzahl der unmittelbaren Spezialisierungen einer Klasse *C*."
Gezählt werden ausschließlich die unmittelbaren Kinder der Klasse. Diese liegen in der Vererbungshierarchie auf Ebene DIT©+1.

Beispiel



Für die Basisklasse *A* in dieser Abbildung gilt:
NOC(*A*) = 2
Analog ergeben sich die folgenden Werte für die verbleibenden Klassen:
NOC© = 2
NOC(*D*) = 1
NOC(*X*) = NOC(*Y*) = NOC(*Z*) = 0

Eigenschaften des Maßes

- Je größer die Zahl der direkten Nachfolger, um so mehr Wiederverwendung durch Vererbung liegt vor.
- Im Gegensatz dazu ist bei steigendem NOC-Wert die Abstraktion zu der Klasse inadäquat.
- Möglicherweise hat eine Klasse mit vielen Kindern einen starken Einfluß auf das Design und muß evtl. umfangreicher getestet werden.

Kommentar

Die Idee zu NOC besteht darin, die Fehlerfortpflanzung einer Klasse auf ihre Kinder zu bewerten.
NOC arbeitet zwar genau genommen auf der Vererbungshierarchie, betrachtet dabei jedoch nur die Ebene unter der gegebenen Klasse. Damit wird NOC als ein Maß eingeordnet, welches ausschließlich auf der Klassenebene arbeitet.

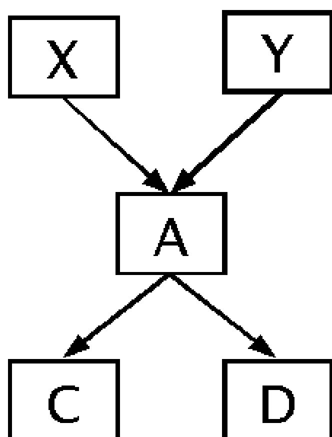
CBO

“Coupling Between Objects”

Definition

CBO© = „Anzahl der Klassen, mit denen die Klasse *C* gekoppelt ist.“
Zwei Klassen sind miteinander gekoppelt, wenn Methoden der einen Klasse Methoden oder Instanzvariablen der anderen Klasse aufrufen bzw. verwenden.
CBO© zählt sowohl die Klassen, die von *C* benutzt werden, als auch die Klassen, die *C* benutzen.

Beispiel



Im Falle dieser Abbildung gilt für die Klasse *A*:
CBO(*A*) = 4

Eigenschaften des Maßes

- Ein Kopplungmaß gibt Aufschluß über die Komplexität des Tests der Systemteile.
- Exzessive Kopplung zwischen Klassen widerspricht dem Konzept der Modularisierung und verhindert Wiederverwendung.
- Je mehr die Klassen eines Systems gekoppelt sind, um so empfindlicher reagiert es auf Änderungen, wodurch der Wartungsaufwand steigt.

Kommentar

Dieses Maß scheint zunächst auf der Aggregationsebene tätig zu sein. Da es sich jedoch immer nur auf eine spezielle Klasse und deren Einbindung in das System bezieht, wird CBO der Klassenebene zugeordnet.

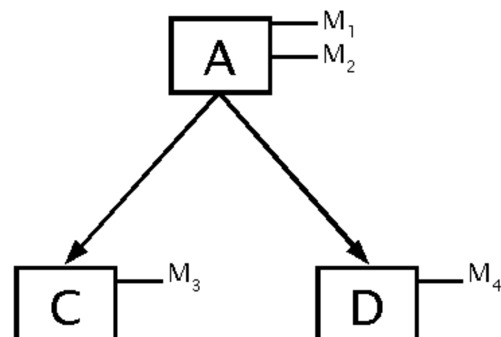
RFC

“Response For a Class”

Definition

RFC© = „Anzahl der Methoden, die potentiell ausgeführt werden können, wenn ein Objekt der Klasse *C* auf eine eingegangene Nachricht reagiert.“
Zu der Menge (auch „Response Set“ genannt) gehören alle Methoden, die direkt aufgerufen werden können. Dazu gehören selbstverständlich auch Methoden, die nicht in der Klasse selbst liegen, sondern durch Kopplung mit anderen Klassen erreichbar sind.

Beispiel



In dieser Abbildung benutzt die Klasse *A* die Klassen *C* und *D*. Angenommen, Methode M_1 ruft M_4 und Methode M_2 ruft M_3 auf. Damit ist das Response Set für *A*
$RS_A=\{M_1, M_2, M_3, M_4\}$
und entsprechend
RFC(*A*) = $|RS_A|$ = 4

Eigenschaften des Maßes

- Je größer die Zahl der Methoden, die aufgerufen werden können, um so komplexer ist die Klasse.
- RFC gibt eine obere Schranke für die Anzahl der Testfälle an, die durch die möglichen Methodenaufrufe notwendig werden.
- Wenn eine große Anzahl Methoden durch eine Nachricht angestoßen werden kann, so wird Test und Debugging komplizierter.

Kommentar

Die Methoden aus dem Response Set können durch Nachrichten an die Objekte der Klasse ausgelöst werden. Auch hier wird wieder ersichtlich, daß RFC der Klassenebene zugeordnet werden muß, da (analog zu CBO) immer die Abhängigkeiten einer Klasse betrachtet werden. Auch hier spielt die Fehlerfortpflanzung eine Rolle, diesmal nicht in der Vererbungsstruktur, sondern auf der Aggregationsebene bezogen auf Methoden.

LCOM

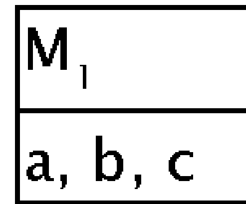
“Lack of Cohesion in Methods”

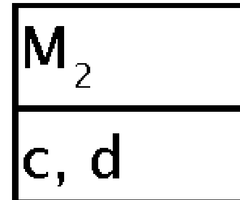
Definition

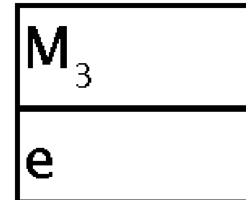
LCOM© = „Anzahl der Paare von Methoden der Klasse *C* ohne gemeinsame Instanzvariablen minus der Anzahl der Paare von Methoden der Klasse *C* mit gemeinsamen Instanzvariablen.“
Das Ergebnis wird auf 0 gesetzt, sofern die obige Subtraktion negativ ist.

Beispiel

Klasse C







Die in dieser Abbildung gezeigten Methoden M_1, M_2 und M_3 der Klasse *C* operieren auf den Instanzvariablen {*a*, *b*, *c*}, {*c*, *d*} und {*e*}. Damit ergibt sich bei der Bildung der paarweisen Schnittmenge:

$$M_1 \cap M_2 = \{c\}$$

$$M_1 \cap M_3 = \emptyset$$

$$M_2 \cap M_3 = \emptyset$$

Damit ergibt sich ein Meßwert von
LCOM © = 2 - 1 = 1

Eigenschaften des Maßes

- Kohäsion von Methoden in einer Klasse ist wünschenswert, weil dadurch die Kapselung in den Objekten gefördert wird.
- Fehlender Zusammenhang bedeutet, die Klasse sollte möglicherweise aufgeteilt werden.
- Geringe oder nicht vorhandene Kohäsion deutet auf Fehler im Design hin.
- Geringer Zusammenhang erhöht die Komplexität und fördert damit die Fehlerwahrscheinlichkeit bei der Entwicklung.

Kommentar

LCOM ermöglicht es, die Kapselung einer Klasse zu überprüfen. Stellt sich zum Beispiel heraus, daß viele Methoden auf disjunkten Attributmengen arbeiten, so deutet dies auf eine schlechte Kapselung der Objekte hin.
Die Definition von LCOM wird diesem Anliegen jedoch nicht voll gerecht, da das Ergebnis nicht unterscheidet, ob keine Methodenpaare mit disjunkter Attributmenge vorhanden sind, oder ob die Differenz tatsächlich gleich Null ist.

Metrikenkataloge

Die hier vorgestellten und darüber hinaus viele andere Metriken werden in der Literatur häufig in Katalogen zusammengefaßt, um als Basis für die Messung von Software zu dienen. Dies ist notwendig, um ein möglichst einheitliches und hinsichtlich der gemessenen Eigenschaften sinnvolles Ergebnis zu erhalten, da es in der Gesamtsicht eines Softwareprojektes kaum Sinn machen wird, immer nur einzelne Maße des Projektes zu überprüfen. Eine Auswahl dieser Kataloge wird im folgenden kurz eingeführt.

MOOSE

Die bereits vorgestellten sechs objektorientierten Metriken stellen ihrerseits die minimale Basis eines Metrikenkataloges dar. Sie wurden von Kemerer ([KEM93]) zur „Metrics Suite for OO-Development“ zusammengefaßt.
Es handelt sich um eine ausgewogene Mischung von Metriken, die sowohl die vorhandenen Klassen selbst als auch die Beziehungen zwischen den Klassen (Vererbung, Aggregation) betrachten.

Sharble/Cohen

Die Metrics-Suit „MOOSE“ wird mit zusätzlichen Maßen verfeinert:
- WAC: weighted attributes per class
- NOT: number of tramps
- VOD: violations of the law of the Demeter

Li/Henry

Im Gegensatz zu Sharble/Cohen wird „MOOSE“ von Li/Henry folgendermaßen erweitert:
- MPC: number of send-statements defined in a class
- DAC: number of ADT’s defined in a class
- NOM: number of logical methods
- SIZE1: number of semicolons in a class
- SIZE2: number of methods plus number of attributes

Grundlagen und Validation

Um mit einem Maß Aussagen über Software treffen zu können, sollte das eingesetzte Maß den meßtheoretischen Grundlagen genügen. Dies ist – wie sich leicht zeigen läßt – für die konventionellen Metriken der Fall. Die objektorientierten Metriken stellen auch hier eine Ausnahme dar. Sie können nicht mehr mit der Maßtheorie erfaßt werden, somit müssen neue Strukturen zur theoretischen Fundierung gefunden werden ([FET95]). Dazu ist es zunächst notwendig, geeignete Verknüpfungsmechanismen zu finden. Diese Mechanis men können in die drei Ebenen
a. Methode
b. Klasse
c. Aggregation
unterteilt werden.
Methoden können verknüpft werden, indem sie entweder hintereinander oder alternativ ausgeführt werden. Klassen können demgegenüber Vereinigt oder Geschnitten werden. Bei der Aggregation werden zwei Klassen von einer (übergeordneten) dritten Klasse benutzt.
Da bei diesen Kombinationsregeln die Additivität nicht mehr gegeben ist, muß zur theoretischen Validierung eine modifizierte Glaubensfunktion eingesetzt werden. Detaillierte Ausführungen dazu finden sich in [FET95].
Als Ergebnis der theoretischen Untersuchungen läßt sich festhalten, daß die OO-Maße tatsächlich nur Maße und keine Metriken im mathematischen Sinne sind.
Gleichfalls nutzt es wenig, wenn der meßtheoretische Hintergrund eines Maßes fundiert wird, ohne die mit dem Maß verknüpfte Aussage zu validieren. Eine der wenigen Arbeiten hierzu stellt [BAS95] dar. In diesem Projekt wurden die sechs in [CHI94] eingeführten und oben beschriebenen Maße auf deren Tauglichkeit hinsichtlich der Fehlerwahrscheinlichkeit untersucht. Dabei stellten sich für die einzelnen Metriken folgende Ergebnisse heraus:
- WMC zeigte seine Signifikanz vor allem bei neuen und stark modifizierten Klassen und Benutzerschnittstellen (GUI / TUI). Dies liegt daran, das die interne Komplexität keinen großen Einfluß hat, wenn eine Klasse mit wenigen oder sehr geringen Änderungen wiederverwendet wird.
- DIT zeigte im gesamten einen starken Einfluß in Bezug auf die Fehlerwahrscheinlichkeit, der - analog zu WMC - bei neuen oder stark veränderten Klassen weiter ansteigt.
- RFC verhält sich vollkommen analog zu DIT.
- NOC zeigte einen den Erwartungen entgegengesetzes Verhalten. Je größer die Zahl der direkten Nachfolger, um so weniger Fehler wurden

gefunden. Dieser Sachverhalt ergibt sich aus der Kombination, daß die meisten Klassen nur ein Kind haben und daß die meisten Klassen mit (relativ) hohem NOC-Wert dazu tendieren, nur in Nuancen verändert zu werden.

- LCOM kann als krasser Ausreißer in diesem Metrikenkatalog betrachtet werden, da ein Einfluß auf die zu erwartenden Fehler in keiner Weise nachgewiesen werden konnte. Begründet wird dies mit einer ungeschickten Definition dieses Maßes, denn wenn mehr (Methoden-) Paare gemeinsame Variablen benutzen als nicht, wird LCOM=0 gesetzt. Dadurch kann die eigentliche Auswirkung auf die Kohäsion nicht erfaßt werden.
- CBO zeigt wie DIT und RFC eine Signifikanz über die gesamte Systembreite.

Durch die „Ausreißer“ NOC und vor allem LCOM wird deutlich, wie wichtig eine adäquate experimentelle Validierung der eingesetzten Maße notwendig ist. Um so verwunderter muß man in diesem Bereich feststellen, daß geeignete Arbeiten zu diesem Themenkomplex praktisch nicht vorhanden sind.

Vergleichbarkeit der Metriken

Die Frage nach der „besseren“ Entwicklungstechnik und damit natürlich auch die Suche nach einer geeigneten Programmiersprache wirft unmittelbar die Problematik der Vergleichbarkeit der eingesetzten Metriken auf. Vor dieser für den Entwickler wesentlichen Fragestellung gilt es, eine noch viel grundsätzlichere Annahme zu validieren: „Objektorientierung fördert die Produktivität und Qualität eines Softwareprojektes.“
Aufgrund der Defizite im OO-Bereich scheiden die konventionellen Metriken wie LOC und MCC von vorne herein aus. Gleiches gilt für die objektorientierten Metriken und die zugehörigen Kataloge. So eignet sich beispielsweise MOOSE zwar durchaus dazu, OO-Projekte miteinander zu vergleichen, aber eine Ausdehnung auf konventionelle Systeme kann nicht erfolgen.
Die einzig sinnvolle Möglichkeit, verschiedene Projekte zu vergleichen, stellt zur Zeit somit die Function-Point Metrik dar. Solange es noch nicht möglich ist, OO-Meßwerte in angemessene konventionelle Meßwerte zu transformieren, bleibt sie auch weiterhin die Wahl, wenn verschiedene Entwicklungstechnologien verglichen werden müssen.
Die zum Teil massiven Unterschiede, die Messungen mit verschiedenen Metriken aufweisen, werden in der folgenden tabellarischen Zusammenfassung einer Studie sichtbar:

Language	Defect Potential	Defect Removal Efficiency	Delivered Defects	Delivered Defects per KLOC	Delivered Defects per FP
Assembly	8635	90,9%	786	2,09	0,52
C	3812	89,1%	415	2,01	0,29
Chill	2726	87,5%	342	2,26	0,23
Pascal	2247	86,7%	295	2,50	0,20
ADA93	1775	85,5%	258	2,77	0,17
ADA9X	1397	83,5%	230	2,94	0,15
C++	1092	80,9%	208	5,47	0,14
Smalltalk	959	79,4%	198	7,07	0,13

Dabei stellt das „Defect Potential“ alle Fehler in der Software dar. Die scheinbar paradoxe abnehmende Effizienz der Fehlerbeseitigung vor der Auslieferung begründet sich darin, daß mit abnehmenden Fehlern im Code die Fehler im Design nicht wesentlich weniger werden, also im Verhältnis zunehmen. Da diese Fehler schwerer zu beseitigen sind, sinkt die Fehlerbehebungsrate.

Sprachspezifische Konstrukte

Die Eigenheiten verschiedener Programmiersprachen stellen die objektorientierten Metriken vor Probleme, die bislang noch nicht gelöst wurden. So gibt es zur Zeit keine geeigneten Maße, um die in der Entwicklung mit C++ üblichen Templates, Friend-Classes und der gleichen sachgerecht zu berücksichtigen, von Exeption-Handling ganz zu schweigen.
Es ist folglich notwendig, die bestehenden Metriken weiter zu entwickeln, um diese Techniken in die Softwaremessung mit einbeziehen zu können, damit deren Nutzen validiert und vorhergesagt werden kann.

OO-Metriken für Entwurf und Dokumentation

Wie schon während der Behandlung der OO-Metriken angedeutet, lassen sich die Maße sofort oder nach geringer Modifikation für die Messung eines Entwurfes verwenden. Auch die Function-Point Metrik bietet sich für die Abschätzung eines Entwurfes an, da sie von der möglichen Implementierung abstrahiert.
Bei der Dokumentation müssen die heute üblichen OO-Metriken offensichtlich außen vor bleiben. Es macht wenig Sinn, einen Text oder einen Graphen auf objektorientierte Eigenschaften zu untersuchen, von Vererbungshierarchien und Aggregationen dargestellter Graphen abgesehen. Es wird deutlich, daß für diese Zwecke die bisher verwendeten Metriken (Textmessung, etc.) weiter benutzt werden müssen.

Literatur

[ABR94] Abreu, Fernando B.: *Candidate metrics for object-oriented software within a taxonomy framework*; Journal of Systems and Software, Vol. 26, 1994, S. 87-96
[BAS95] Basili, Victor R.; Briand, Lionel; Melo, Walcèo L.: A validation of object-oriented design metrics as quality indicators; Technical Report, University of Maryland, April 1995
[CHI94] Chidamber, Shyam R.; Kemerer, Chris E.: *A metrics suite for object oriented Design*; IEEE Transactions on Software Engineering, Vol. 20, Nr. 6, Juni 1994, S. 476-493
[DRE89] Dreger, Brian J.: *Function point analysis*; Prentice Hall, 1989
[FET95] Fetcke, Thomas: *Softwaremetriken bei objektorientierter Programmierung*; GMD-Studien 259, April 1995, Gesellschaft für Mathematik und Datenverarbeitung mbH
[IFP94] Garmus, David (Editor): *IFPUG Counting practice manual, Release 4.0*; International Function Point User Group (IFPUG), April 1994
[KEM93] Kemerer, Chris E.: MOOSE: *Metrics for Object Oriented System Environments*; Proceedings of the ASM 93 Conference
[MCC76] McCabe, Thomas: *A complexity measure*; IEEE Transactions on Software Engineering, Vol. SE-1, Nr. 3, 1976, S. 312-327
[MOR89] Moreau, Dennis R.; Dominick, Wayne D.: *Object-oriented graphical information systems: research plan and evaluation metrics*; The Journal of Systems and Software, Vol. 10, 1989, S. 23-28
[LI93] Li, Wei; Henry, Sallie: *Object-oriented metrics that predict maintainability*; Journal of System and Software, Vol. 23, Nr. 2, November 1993, S. 111-122
[LOR94] Lorenz, Mark; Kidd, Jeff: *Object-oriented Software Metrics*; Prentice-Hall, 1994
[TEG92] Tegarden, David P.; Sheetz, Steven D.; Monarchi, David E.: *Effectivness of traditional software metrics for object-oriented systems*; Proceedings of the HICSS-92, San Diego, 1992

Copyright © Oliver Feuser 1997 – All Rights Reserved.