Objektbasierte Maße

Stephan Hagendorf
Institut für Informatik,
Freie Universität Berlin
stephanhagendorf@web.de

Kurzfassung

Wenn man die zentrale Rolle der Softwareentwicklung bei der Anwendung von Informationstechnologie in Betracht zieht, wundert es nicht, dass sich Entwicklungsleiter oder auch Projektleiter immer mehr auf die Verbesserung auf dem Gebiet der Softwareentwicklung konzentrieren. Diese neue Anforderung resultiert in der Bestimmung einer Anzahl von neuen und/oder fortgeschrittenen Ansätzen in der Softwareentwicklung, wobei die wahrscheinlich bekannteste die Objektorientierung ist. Hinzu kommt, dass der Fokus auf die Prozessverbesserung bei der Entwicklung von Software den Bedarf an Softwaremaßen, mit denen man diese Prozesse „managen" kann, erhöht. Der Bedarf für derartige Maße kann unter Umständen sehr akut werden, etwa wenn eine Organisation eine neue Technologie übernimmt, für die es bisher keine etablierten Praktiken gibt. Maße aus bisherigen Forschungen, obwohl sie zum Verständnis des Feldes des Softwareentwicklungsprozesses beigetragen haben, waren generell ernsthaften Kritiken ausgesetzt was besonders den Mangel an theoretischer Fundiertheit angeht. In der Arbeit von Chidamber und Kemerer 1994 wurden 6 Entwurfsmaße geschaffen und analytisch gegen Weyuker's „set of measurement principles" evaluiert. Weiterhin wurde ein Tool entwickelt und implementiert um empirische Proben von realer Software an zwei Entwicklungsstätten zu sammeln und die Möglichkeiten dieser Maße zu zeigen. Außerdem sollen Wege vorgeschlagen werden wie Entwickler sie zur Verbesserung des Softwareentwicklungsprozesses nutzen können. Da diese Studie auch teilweise heftigen Kritiken ausgesetzt war, werden im Anschluss weitergehende empirische Studien sowie einige Analysen umrissen.

1. Einleitung

Es ist auf breiter Front anerkannt, dass die Fähigkeit, Prozesse zu messen, einen wichtigen Faktor in der Prozessverbesserung darstellt. Wenn man die zentrale Rolle der Softwareentwicklung bei der Anwendung von Informationstechnologie in Betracht zieht, wundert es nicht, dass sich leitende Entwickler und Manager immer mehr auf die Verbesserung auf dem Gebiet der Softwareentwicklung konzentrieren. Diese Konzentration hat zwei Effekte: Erstens resultiert aus dieser Anforderung eine Anzahl von neuen und/oder fortgeschrittenen Ansätzen in der Softwareentwicklung, wobei die wahrscheinlich bekannteste die Objektorientierung ist. Hinzu kommt der Fakt, dass die objektorientierte Softwareentwicklung seit Anfang der 90er Jahre rasanten Zuwachs erfährt und immer mehr Unternehmen objektorientierte Techniken und Entwicklungsmethoden anwenden. Dementsprechend ergibt sich auch der Bedarf an speziell objektorientierten Richtlinien, Empfehlungen und Entwurfsprinzipien, an die sich Softwareentwickler halten sollten, aber auch Kontrollmechanismen in Form von Softwaremaßen, an denen die Einhaltung dieser Richtlinien „gemessen" und geprüft und somit die Qualität des Softwareproduktes sichergestellt werden kann. Zweitens hat der Fokus auf „process improvement" den Bedarf an Softwaremaßen oder Metriken, mit denen man diese Prozesse „managen" kann, erhöht. Der Bedarf für derartige Maße kann unter Umständen sehr akut werden, etwa wenn eine Organisation eine neue Technologie übernimmt, für die es bisher keine etablierten Praktiken gibt. Die Forschungen von Chidamber & Kemerer behandeln diese Bedürfnisse durch das Entwickeln und Implementieren eines neuen Satzes an Maßen für den objektorientierten Entwurf. Maße aus bisherigen Forschungen, obwohl sie zum Verständnis des Feldes des Softwareentwicklungsprozesses beigetragen haben, waren generell ernsthafter Kritik ausgesetzt. Dies beinhaltet mangelnde theoretische Basis [39], Mangel an notwendigen Messeigenschaften [40], sie waren ungenügend verallgemeinert oder zu abhängig von der Implementationstechnologie [41] und zu schwer zu extrahieren. Wenn man Wand und Weber folgt, war die theoretische Grundlage für die vorgeschlagenen Maße „the ontology of Bunge" [42], [43], [44]. Sechs Entwurf-Maße wurden entwickelt und analytisch gegen einen vorgeschlagenen Satz an Messprinzipien evaluiert. Es wurden Daten mit Hilfe eines eigens dafür entwickelten Tools bei zwei verschiedenen Software-Entwicklungsstätten gesammelt, um die Eigenschaften dieser Maße zu demonstrieren und um Einsatzmöglichkeit für „Manager" in der Prozessverbesserung vorzuschlagen. Dieser Prozess der empirischen Validierung von objektorientierten Softwaremaßen dauert bis heute an.

2. Theoretische Vorüberlegungen

Um im weiteren Verlauf eventuelle Unklarheiten zu vermeiden, sind nachfolgend einige Begriffsdefinitionen gegeben, die später direkt oder indirekt in der Arbeit auftauchen werden.

2.1. Begriffsdefinitionen

2.1.1. Kopplung

Die Kopplung (coupling) einer Einheit beschreibt die Wechselwirkungen, die sie zu anderen Einheiten unterhält und die damit verbundenen Abhängigkeiten. Die zentrale Frage bei der Bewertung der Kopplung ist, wie viel man über ein oder mehrere Module wissen muss, um ein Modul zu warten oder zu verstehen. Dementsprechend herrscht ein sehr starker Zusammenhang zwischen der Kopplung zwischen Modulen eines Softwareproduktes und Qualitätseigenschaften wie Zuverlässigkeit, Wart- und Änderbarkeit, Wiederverwendbarkeit und Verständlichkeit. Generell ist beim Entwurf von Systemen wichtig, unnötige Kopplungen zu vermeiden und den Grad der notwendigen Kopplungen so gering wie möglich zu halten. In bezug auf die objektorientierte Programmierung sollte berücksichtigt werden, dass die Vererbung neue Formen der Kopplung einführt. Andererseits erscheint intuitiv, dass sich der objektorientierte Ansatz insgesamt positiv
auf die Kopplung auswirkt. Diese beiden Tatsachen zeigen, dass eine Bewertung der Kopplung nach herkömmlichen Softwaremaßen beim OO-Ansatz nicht erfolgreich sein kann, so dass der Bedarf an spezifischen OO-Maßen offensichtlich wird.

2.1.2. Bindung

Die Bindung oder auch Kohäsion (cohesion) beschreibt als semantische Eigenschaft qualitativ, wie gut die zu einer Einheit zusammengefassten Elemente der Repräsentation eines gemeinsamen inhaltlichen Konzepts dienen und wie gut die Einheit als Abstraktion geeignet ist. Eine Klasse weist z.B. dann eine hohe Bindung auf, wenn sie ein klar definiertes einheitliches Konzept minimal aber vollständig repräsentiert. Das Ziel, die Bindung einer Methode, Klasse oder Vererbungshierarchie zu erfassen, entpuppt sich jedoch als äußerst schwierig. Dies liegt eindeutig daran, dass sie als semantische Eigenschaft kaum formal und objektiv erfasst werden kann. Dennoch gibt es einige Ansätze, die später noch erwähnt werden.

2.2. Warum Objektbasierte Maße?

Die Antwort auf diese Frage ist kurz und nicht unumstritten: „Weil man spezifisch objektorientierte Konzepte berücksichtigen muss.“, so jedenfalls die nicht unumstrittene Behauptung von anerkannten Forschern, wie etwa Chidamber und Kemerer. Die folgenden Konzepte sind charakteristisch für objektorientierte Software und teilweise sehr verschieden zu den gleichlautenden Begriffen in traditioneller Software. Das legt zunächst die Vermutung nahe, dass wenn man diese Konzepte nicht ausreichend bei der Suche nach geeigneten Maßen berücksichtigt, jedes auf anderem Wege gefundene Maß nicht genau oder nicht repräsentativ wäre.
Die betrachteten Konzepte sind im Einzelnen:

2.2.1. Lokalität

Lokalität ist das Platzieren von „Dingen“ in geringer (physischer) Nähe zueinander. Funktionale Dekomposition ist das Anordnen von Information um Funktionen, Datenlokalität ist das Anordnen von Information um Daten, wohingegen im objektorientierten Ansatz Lokalität das Anordnen von Daten um Objekte ist. In der konventionellen

Softwareentwicklung basiert die Lokalität auf Funktionalität, daher hat sich das Messen traditionellerweise auf Funktionen und Funktionalität fokussiert. Einheiten der Software waren funktionale Einheiten, deshalb konzentrierten sich die Maße auf Komponentenbeziehungen, die funktionale Zwischenbeziehungen betonen, z.B. Modulkopplung. Lokalität in objektorientierter Software, jedenfalls basiert auf Objekten. Das heißt: Die Maßfindung und die Messung müssen das "Objekt" als Basiseinheit der Software sehen. Das Problem hierbei ist, dass innerhalb eines Systems von Objekten die Lokalität zwischen Funktionalität und Objekten keine Eins-zu-eins-Beziehung ist. Zum Beispiel könnte eine Funktion mehrere Objekte involvieren und ein Objekt könnte viele Funktionen bereitstellen.

2.2.2. Kapselung

Kapselung ist das "Zusammenschnüren" einer Menge von Entitäten. Man unterscheidet mehrere Arten der Kapselung. Es gibt die so genannte Low-level–Kapselung, etwa Arrays, Prozeduren oder Subroutinen. Objektorientierte Programmiersprachen erlauben "Higher-level"-Kapselung, z.B. Klassen in C++ und Java. Kapselung hat zwei große Einflüsse auf das Messen: Zum einen ist die Basiseinheit nicht mehr das Unterprogramm, sondern das Objekt und zum anderen muss man das Denken in bezug auf die Charakterisierung und Berechnung von Systemen umstellen.

2.2.3. Information Hiding

Information Hiding ist das Zurückhalten (oder Verstecken) von Details. Die generelle Idee ist, dass man nur die Information zeigt, die nötig ist, um ein bestimmtes Ziel zu erfüllen. Es gibt Abstufungen des Information Hiding, die von teilweiser "verbotener" Sichtbarkeit bis hin zur totalen Nichtsichtbarkeit reichen. Kapselung und Information Hiding sind nicht das selbe, z.B. kann etwas gekapselt aber dennoch total sichtbar sein. Information Hiding spielt eine direkte Rolle in Maßen zur Objekt-Kopplung.

2.2.4. Vererbung

Vererbung ist ein Mechanismus, bei dem ein Objekt Charakteristiken von einem anderen oder mehreren anderen bekommt. Einige objektorientierte Sprachen unterstützen einfache Vererbung, einige unterstützen Mehrfachvererbung. Viele objektorientierte Maße basieren auf Vererbung, z.B.: Anzahl der Kinder, Anzahl der Eltern oder Tiefe einer Klasse im Vererbungsbaum.

2.2.5. Wiederverwendung

In der objektorientierten Entwicklung ist Wiederverwendung eine zentrale Frage. Dies äußert sich in Wiederverwendung von Bibliotheken oder Frameworks, Wiederverwendung durch Vererbung und Wiederverwendung auf Metacode-Ebene, etwa Patterns oder Business Objects. Wiederverwendung verändert also den Entwicklungsprozess, d.h. man achtet, neben den üblichen Aufgaben in der Softwareentwicklung, auf das Erstellen von wiederverwendbaren Komponenten und auf das Finden von Komponenten, die man wiederverwenden kann, bevor man eine neue Komponente entwickelt.

2.3. Klassifikation

OO-Maße werden unter anderem nach dem Zeitpunkt der Anwendung klassifiziert, z.B. Analyse, Entwurf, Implementierung, Test oder Wartung. Auch das Softwareprodukt, auf das sich das Produkt bezieht, liefert ein Unterscheidungskriterium. Dabei werden unter Softwareprodukten Entwürfe, Dokumentationen, Use-Cases, Quellcode usw. verstanden. Von entscheidender Bedeutung für die Praxis ist die Frage, ob ein Maß in dem Sinne automatisierbar ist, als dass es von einem Tool bestimmt werden kann. Ein weiteres Kriterium ist, ob das Maß einfach oder zusammengesetzt ist. Ein Maß wird als zusammengesetzt bezeichnet, wenn es mehrere einfache Maße kombiniert, um ein abstraktes Qualitätsmerkmal zu charakterisieren. Im Rahmen dieser Arbeit werden jedoch nur einfache und automatisierbare Maße betrachtet, die sich auf den Entwurf als Entwicklungsphase und Softwareprodukt

beziehen. Ein wichtiges Unterscheidungsmerkmal ist auch die Abstraktionsebene, die von dem Maß betrachtet wird. Folgende Ebenen werden unterschieden:
Methodenebene, Klassenebene, Vererbungshierarchie und (Sub)Systemebene.
Dabei sollten die Begriffe der Methode, der Klasse und der Vererbungshierarchie dem Leser geläufig sein. Als Subsystem wird im folgenden ein Teil des Systems (z.B. eine Gruppe von Klassen) bezeichnet, das eine Einheit bildet und einem konkret definierten Zweck dient. Von zentraler Bedeutung sind jedoch die Eigenschaften bzw. Qualitätsmerkmale, die das Maß quantitativ zu erfassen versucht. In dieser Arbeit werden Maße für den objektorientierten Entwurf behandelt, da sie bezüglich ihrer Nützlichkeit im Softwareentwicklungsprozess früh genug ansetzen, um so viele Defekte wie möglich von vorneherein zu vermeiden und dort bereits recht umfangreiche Daten über die Software zur Verfügung stehen.

2.4. Gefahren bei der Interpretation

Bei der Suche nach geeigneten Maßen für die unterschiedlichen Aspekte ist sehr vorsichtig vorzugehen. Denn nicht immer sind Beobachtungen und die daraus gezogenen Schlüsse exakt. Ein Beispiel hierfür ist sind die Schlüsse einiger Wissenschaftler im Vorfeld einer Studie über die optimale Größe einer Klasse von Melo et. al. [44]. Bis dato herrschte unter diesen Wissenschaftlern die Meinung, es gäbe eine optimale Größe für eine Klasse. Das heißt eine bestimmte Größe bzw. ein bestimmter Umfang (z.B. Anzahl der ausführbaren Zeilen) ist optimal in Hinsicht auf ihre Anfälligkeit für Defekte. Wenn man sich vorstellt, dass mit zunehmender Größe einer Klasse auch deren Komplexität zunimmt, scheint dies intuitiv richtig zu sein. Die Theorie von der optimalen Klassengröße besagt jedoch zudem, dass unterhalb dieser Größe die Defektanzahl im Verhältnis zur Größe wieder zunimmt, je kleiner die Klasse ist. Wäre dies tatsächlich zutreffend, entstünden daraus enorme Konsequenzen für den Softwareentwicklungsprozess, denn man könnte nun alle Klassen, oder zumindest so viele wie möglich, auf den optimalen Wert ausrichten, um die Defektrate zu minimieren. Die Theorie hat sich lange gehalten, weil immer wieder Beobachtungen gemacht wurden, die sie unterstützten. Sie war so glaubhaft, dass bereits versucht wurde, ein kognitives Modell dafür zu finden. Man sollte aber beachten, dass in den Auswertungen der betreffenden Studien stets Defektrate und Größe gegenübergestellt wurden. Nun kann man zeigen, dass, wann immer eine Variable x ihrem Reziproken 1/x gegenübergestellt ist, eine negative Verbindung resultiert, solange das Verhältnis von Größe und Defekten ungefähr linear ist, so die Analyse von Melo et. al.
Würde man dies auf eine medizinische Studie übertragen, etwa der Vergleich von Sterblichkeit und Menge an täglich gerauchten Zigaretten, so würde man im Hinblick auf Sterblichkeit eine optimale Anzahl von Zigaretten herausbekommen! [44]

3. Chidamber und Kemerer

Anfang der 90er Jahre stellten die Wissenschaftler Shyam R. Chidamber und Chris F. Kemerer einen Satz von Entwurfsmaßen auf und lieferten 1994 entsprechende empirische Daten dazu, die die Eigenschaften dieser Maße zeigen sollten. Um eine gewisse mathematische Fundiertheit in die Maßfindung einzubringen, war es nötig eine Menge von formalen Kriterien zu besitzen, mit denen man vorgeschlagene Maße evaluieren kann. Weyuker [4] hat eine formale Liste von wünschenswerten Kriterien für Softwaremaße aufgestellt. Obwohl Zuse [11] kritisiert hat, dass diese Kriterien nicht konform mit den Prinzipien der Skalen gehen, wurden sie von Chidamber und Kemerer zur analytischen Evaluierung benutzt (wohl aus Gründen der Bekanntheit). Nachfolgend sind die Eigenschaften umrissen:

3.1. gewünschte Eigenschaften

3.1.1. hinreichende Feinheit

Zu einer gegebenen Klasse P und einem Maß μ existiert immer eine weitere Klasse Q so dass $\mu(P) \neq \mu(Q)$. Das impliziert, dass nicht jede Klasse den selben Messwert hat, sonst würde das Maß ja seine Messeigenschaft verlieren.

3.1.2. Nichteinzigartigkeit

Es können bestimmte Klassen P und Q derart existieren, so dass $\mu(P) = \mu(Q)$. Dies bedeutet, dass zwei Klassen den selben Messwert haben können, also gleich komplex sind.

3.1.3. Entwurfsdetails sind wichtig

Aus zwei gegebenen Klassenentwürfen P und Q, die die selbe Funktionalität liefern, folgt nicht $\mu(P) = \mu(Q)$. Der Entwurf der Klasse muss den Messwert der Klasse verändern (können).

3.1.4. Monotonität

Für alle Klassen P und Q muss gelten: $\mu(P) \leq \mu(P+Q)$ und $\mu(Q) \leq \mu(P+Q)$, wobei P+Q die Kombination von P und Q impliziert. Das heißt, dass das Maß für eine Kombination niemals kleiner sein kann als das selbe Maß auf eine der kombinierten Klassen angewendet.

3.1.5. Nichtäquivalenz von Interaktion

Es existieren drei Klassen P, Q und R derart, das aus $\mu(P) = \mu(Q)$ nicht folgt,
dass $\mu(P+R) = \mu(Q+R)$. Das bedeutet, dass die Interaktion zwischen P und R kann anders sein kann als die Interaktion zwischen Q und R, was sich in unterschiedlichen Messwerten äußern muss.

3.1.6. Interaktion vergrößert Komplexität

Es existieren zwei Klassen P und Q derart, dass $\mu(P) + \mu(Q) < \mu(P+Q)$. Das Prinzip hierbei ist, dass wenn zwei Klassen kombiniert sind, sich der Wert des Maßes vergrößert.

Die Maße, die Chidamber und Kemerer gefunden hatten, wurden nun hinsichtlich dieser Kriterien analytisch untersucht, und um sie zu validieren, wurden sie an zwei verschiedenen Softwareentwicklungsstätten in der Industrie mittels eines speziell dafür entwickelten Tool genommen. Die untersuchten Systeme an den Entwicklungsstätten unterschieden sich dabei sowohl in der Art der Anwendung, der Anzahl der beteiligten Klassen, als auch in der verwendeten Programmiersprache (C++, Smalltalk).

3.2. Die CK-Maße

3.2.1. Weighted Methods Per Class (WMC)

Definition: Angenommen eine Klasse C1 definiert die Methoden M1,...,Mn. Ferner sei $c_1,...,c_n$ die jeweilige Komplexität der einzelnen Methoden dann gilt:

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

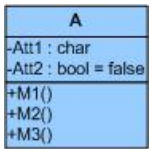


Abbildung 1 – Weighted Methodes per Class

In Abbildung 1 sieht man eine Klasse A mit drei Methoden. Nimmt man einmal eine Komplexität von ‚1' für alle Komplexitäten der einzelnen Methoden an, so ergibt sich für die Klasse A:
WMC(A) = 3.

Was die Komplexität einer Methode ausmacht, sei hier absichtlich nicht spezifischer definiert, um eine möglichst allgemeine Nutzung des Maßes zu erlauben. Es kann so argumentiert werden, dass Entwickler so an die Aufgabe eine Methode zu schreiben herangehen, wie sie es für ein traditionelles Programm tun würden. Daher könnten einige traditionelle statische Komplexitätsmaße passen. Es handelt sich letztendlich um eine Implementationsentscheidung da man sich bisher nicht auf die Anwendung eines existierenden Komplexitätsmaßes geeinigt hat. Jedes Komplexitätsmaß, dass dazu benutzt wird, sollte die Eigenschaft einer Intervallskala besitzen, um die Summation zu erlauben.

Sichten auf das WMC-Maß:
• Die Anzahl der Methoden und die Komplexität dieser Methoden sagt vorher, wieviel Zeit und Aufwand benötigt wird, um die Klasse zu entwickeln und zu warten.
• Je größer die Anzahl der Methoden einer Klasse ist, desto größer sind die möglichen Auswirkungen auf Kindesklassen, weil die Kindesklassen alle in der Klasse definierten Methoden erben werden.
• Klassen mit einer großen Anzahl an Methoden sind wahrscheinlich etwas anwendungsspezifischer, was die Möglichkeit der Wiederverwendung einschränkt..
Analytische Evaluation von Weighted Methods Per Class (WMC)
Chidamber & Kemerer bewiesen hier, dass die oben genannten Messeigenschaften 1 – 5 für das WMC Maß gelten, jedoch nicht Eigenschaft 6. Dies wird später in der Ausführung diskutiert werden.

Interpretation der empirischen Daten:
Die interessanteste Entdeckung bezüglich der Daten ist die gleichartige Verteilung der Maße an den beiden Entwicklungsstätten, trotz der Unterschiede in der Art der Anwendung, den am Entwurf beteiligten Leuten und den verwendeten Programmiersprachen. Dies rührt daher, dass die meisten Klassen dazu tendieren, eine kleine Anzahl von Methoden zu haben, während ein paar „Ausreißer" eine große Anzahl von Methoden deklarieren. Die meisten Klassen in einer Anwendung sind relativ einfach beschaffen, um eine spezifische Abstraktion und Funktionalität zu erreichen. In den „Ausreißerklassen" an Entwicklungsstätte A war etwas Interessantes zu beobachten: Die Klasse mit der maximalen Anzahl an Methoden (106) hatte keine „Kinder" und war an der Wurzel in der Hierarchie, wohingegen eine andere „Ausreißerklasse" mit 87 Methoden 14 Unterklassen eine Gesamtanzahl an „Nachfahren" von 43 hatte. Im ersten Fall haben die Methoden der Klasse keine Wiederverwendung innerhalb der Anwendung und

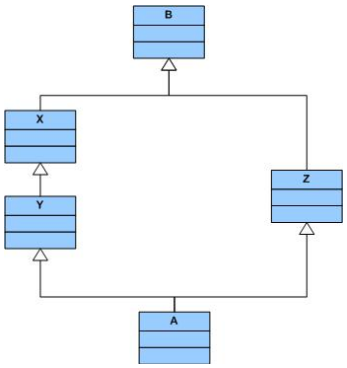


Abbildung 2 Depth in Inheritence Tree

sofern dies keine allgemeine Klasse ist, die über Anwendungen hinweg wiederverwendet wird, ist der Aufwand, den man betreibt um diese Klasse zu entwickeln scheinbar schlecht angelegt. Die Klasse mit den 87 Methoden jedenfalls hat ein beachtliches Wiederverwendungs-potential innerhalb der Anwendung und man sollte ihr besondere Beachtung beim Testen schenken, da die Methoden eine Weitgefächerte Verwendung im System haben können.

3.2.2. Depth of Inheritance Tree (DIT)

Definition: Das DIT-Maß einer Klasse ist definiert als der maximale Weg von der Wurzel der Vererbungshierarchie bis zur betrachteten Klasse.

In der Abbildung 2 gilt für die Klasse A:
DIT(A) = 3, da die maximale Anzahl der Teilwege von der Wurzel, also der Klasse B bis zur betrachteten Klasse A ‚3' beträgt.

Sichten auf das DIT-Maß:
• Je tiefer eine Klasse in der Hierarchie ist, desto größer ist die Anzahl der Methoden, die sie wahrscheinlich erben wird. Daher ist Verhalten schwieriger (komplexer) vorherzusagen.
• Tiefe Vererbungshierarchien tragen zu erhöhter Entwurfskomplexität bei, da mehr Methoden und Klassen involviert sind.
• Je tiefer eine bestimmte Klasse in der Vererbungshierarchie ist, desto größer ist das Wiederverwendungspotential der geerbten Methoden.

analytische Evaluation des DIT-Maßes:
Die Analyse von DIT durch C. & K. ergab, dass die Eigenschaften 1, 2, 3 und 5 gelten. Eigenschaft 4 gilt nur manchmal, während Eigenschaft 6 nicht gilt.

Interpretation der empirischen Daten:
An beiden Entwicklungsstätten waren kleine bis mittlere Werte für DIT zu beobachten. Beide Klassenbibliotheken wurden als „top heavy" eingestuft, d.h. es befanden sich viele Klassen in der Nähe der Wurzel, was vermuten lässt, dass die Entwickler den Vorteil der Wiederverwendung von Methoden durch Vererbung nicht nutzten. Weiterhin waren die maximalen DIT-Werte ziemlich klein (10 oder kleiner) – eine mögliche Erklärung dafür ist, dass die Entwickler dazu tendieren, die Ebenen der Abstraktion auf eine überschaubare Anzahl zu begrenzen. Weiterhin könnten die Entwickler Wiederverwendung durch Vererbung zugunsten der einfachen Verständlichkeit opfern.
Dies zeigt unter anderem auch einen Vorteil, den das Extrahieren von Entwurfsmaßen mit sich bringt, nämlich dass man ein klareres Bild von der Konzeption von Softwaresystemen erhält, unter besonderer Beachtung der Kompromisse im Entwurf. Die Untersuchung einer Klasse an der Entwicklungsstätte A mit einem DIT-Wert von 8 zeigte einen Fall von fortführender Abstraktion eines graphischen Konzepts von Control Panels. Die Klasse selbst

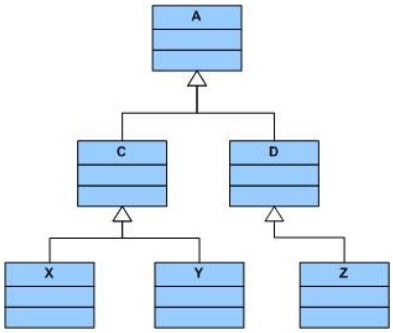


Abbildung 3 – Number of Children

hatte nur 4 Methoden und lediglich lokale Variablen, aber Objekte dieser spezialisierten Klasse brachten es auf eine Gesamtanzahl von 132 durch Vererbung erreichbare Methoden. Diese Klasse zu entwerfen wäre eine recht einfache Aufgabe aber durch die starke Vererbung könnte das Testen komplizierter sein.

3.2.3. Number of Children (NOC)

Definition: NOC definiert die Anzahl der direkten Unterklassen einer Klasse.

In Abbildung 3 ist eine Vererbungshierarchie dargestellt. Für die Klasse A gilt:
NOC(A) = 2, da sie 2 direkte Unterklassen besitzt.

Sichten auf NOC:
• Je größer die Anzahl der direkten Unterklassen, desto größer die Wiederverwendung, da Vererbung eine Form der Wiederverwendung ist.
• Je größer die Anzahl der direkten Unterklassen, desto höher die Wahrscheinlichkeit, dass die Oberklasse ungeeignet abstrahiert ist. Wenn eine Klasse eine große Anzahl von „Kinder“ hat, könnte es sich um einen Fall von Missbrauch des Konzepts des sub-classing handeln.
• Die Anzahl der “Kinder” vermittelt einen Eindruck des möglichen Einflusses, den die Klasse im Entwurf hat. Wenn eine Klasse eine große Anzahl von „Kinder“ hat, könnte für die Methoden in dieser Klasse mehr Testaufwand nötig sein.

Analytische Evaluation von NOC:
Den Analysen von C. & K. zu Folge sind die Eigenschaften 1-5 erfüllt. Eigenschaft 6 ist nicht erfüllt.

Interpretation der empirischen Daten: Auch bezüglich des NOC-Maßes ist die Gleichverteilung an den Entwicklungsstätten interessant (wie bei WMC). Das bedeutet, dass Klassen generell wenige direkte Kinder haben und dass nur einige „Ausreißer“ viele direkte Unterklassen haben.

3.2.4. Coupling between object classes (CBO)

Definition: Der CBO-Wert einer Klasse ist die Anzahl von anderen Klassen mit der die betreffende Klasse gekoppelt ist. Zwei Klassen sind gekoppelt, wenn in der eine Klassen Methoden oder Instanzvariablen der anderen Klasse benutzt werden.

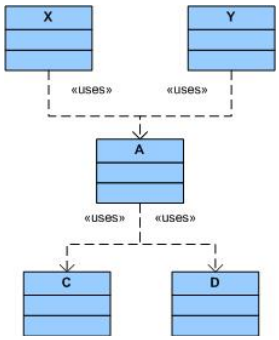


Abbildung 4 - Coupling between Objekts

In Abbildung 4 sieht man eine Klasse A, die einerseits von zwei Klassen X und Y benutzt wird, und andererseits selber zwei Klassen – C und D benutzt, d.h. sie „kommuniziert mit allen vier Klassen.
Daher gilt:
CBO(A) = 4

Sichten auf CBO:
• Je unabhängiger eine Klasse ist, desto einfacher ist sie in einer anderen Anwendung wiederzuverwenden.
• Um die Modularität zu verbessern und Kapselung zu unterstützen, sollten die Kopplungen der Objekte auf ein Minimum begrenzt werden. Je größer die Anzahl der Kopplungen, desto höher ist die Anfälligkeit auf Veränderung in anderen Teilen des Entwurfes, und somit ist die Wartung schwieriger.
• Ein Kopplungsmaß ist nützlich um zu bestimmen wie komplex das Testen verschiedener Teile des Entwurfs wahrscheinlich sein wird. Je größer die Anzahl der Kopplungen, desto rigoroser muss getestet werden.

Analytische Evaluation von CBO:
Das CBO-Maß erfüllt die Eigenschaften 1-5, nicht jedoch Eigenschaft 6.

Interpretation der empirischen Daten:
Es wurde festgestellt, dass die SmallTalk-Anwendung im Gegensatz zur C++ Anwendung relativ hohe mittlere Werte für CBO hatte. Eine mögliche Erklärung dafür sei die unterschiedliche Art der Anwendung aber noch wahrscheinlicher sei, dass dies aus dem Unterschied der beiden Programmiersprachen resultiert. Die trotz alledem generell niedrigen Durchschnittswerte für CBO seien ein Zeichen dafür, dass mindestens 50% der Klassen nicht mit anderen Klassen kommunizieren. Außerdem hätten die meisten Klassen keine Eltern oder Kinder und so könnte der begrenzte Gebrauch der Vererbung auch für niedrige CBO-Werte verantwortlich sein. Eine Untersuchung der „Ausreißerklassen“ an Entwicklungsstätte B ergab, dass Klassen, die verantwortlich für das Kommunizieren mit Schnittstellen waren, hohe CBO-Werte hatten. Diese Klassen tendierten dazu als Verbindungspunkte zwischen zwei oder mehreren Subsystemen innerhalb der Anwendung zu fungieren.

3.2.5. Response For a Class (RFC)

Definition: RFC = |RS|, wobei RS das sogenannte response set ist, welches alle Methoden, die direkt aufgerufen werden können und alle Methoden die durch Kopplung mit anderen Klassen erreichbar sind (eine Ebene nach außen) beinhaltet.

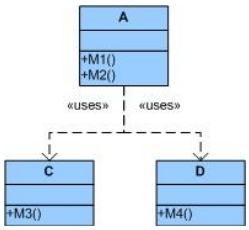


Abbildung 5 – Resonse for a Class

In Abbildung 5 sieht man drei Klassen – A, C und D. Angenommen die Methode M1 der Klasse A benutzt die Methode M3 der Klasse C und die Methode M2 von A benutzt die Methode M4 der Klasse D.
Damit gilt für das Response Set von A:
RSA={M1, M2, M3, M4}
und entsprechend

RFC(A) = |RSA| = 4

Sichten auf das RFC-Maß:
• Wenn eine große Anzahl von Methoden durch eingehende Nachrichten erreichbar ist, wird das Testen und Debugging der Klasse komplizierter weil es vom Tester ein größeres Verständnis fordert.
• Je größer die Anzahl der Methoden, die von einer Klasse erreichbar sind, desto größer die Komplexität der Klasse.
• Ein "worst case" - Wert für mögliche Reaktionen auf eingehende Nachrichten hilft dabei angemessene Testzeit bereitzustellen.

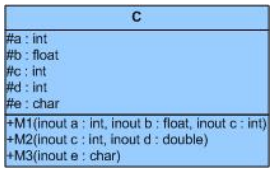
analytische Evaluation:
Wiederum wurden die Eigenschaften 1-5 erfüllt, jedoch nicht Eigenschaft 6.

Interpretation der empirischen Daten
Die empirischen Daten würden die These untermauern, dass einige wenige Klasse verantwortlich für die Ausführung vieler Methoden in der Anwendung sind, weil sie entweder viele Methoden haben (dies scheint der Fall bei Entwicklungsstätte A zu sein) oder sie viele Methoden aufrufen. Die unterschiedlichen Werte für RFC bei den beiden Entwicklungsstätten würden abermals auf die unterschiedlichen Programmiersprachen zurückzuführen sein: Während das relativ starke Einhalten von Objektorientierung viele Methodenaufrufe in Smalltalk mit sich bringe, gäbe der inkrementelle Ansatz der Objektorientierung in C++ den Entwicklern Alter-nativen zum Methodenaufruf durch „message passing".

3.2.6. Lack of Cohesion in Methods (LCOM)

Defintion: Der LCOM-Wert einer Klasse ist die Anzahl der Paare von Methoden in der Klasse ohne gemeinsame Instanzvariablen minus der Anzahl der Paare von Methoden in der Klasse mit gemeinsamen Instanzvariablen.

Abbildung 6 zeigt eine Klasse C. Angenommen die Methoden M1, M2 und M3 der Klasse C operieren auf den Instanzvariablen {a, b, c}, {c, d} und {e}.



Abblidung 6 – Lack of Cohesion in Methods

Dann sind die paarweisen Schnittmengen:

$M1 \cap M2 = \{c\}$

$M1 \cap M3 = \varnothing$

$M2 \cap M3 = \varnothing$

LCOM (C) = 2 - 1 = 1

Sichten auf das LCOM-Maß:

• Die Bindung einer Klasse ist wünschenswert, da sie Kapselung unterstützt

• Der Mangel an Bindung impliziert, dass die Klassen in zwei oder mehrere Unterklassen unterteilt werden sollten.

• Jedes Maß für die „Unvereinbarkeit von Methoden" hilft dabei, Fehler im Klassenentwurf zu identifizieren.

• Eine geringe Bindung impliziert hohe Komplexität und daher eine höhere Wahrscheinlichkeit für auftretende Fehler

analytische Evaluation von LCOM:

LCOM erfüllt die Eigenschaften 1-3 sowie Eigenschaft 5, nicht jedoch Eigenschaft 4 und Eigenschaft 6.

Interpretation der empirischen Daten:

An beiden Entwicklungsstätten waren die Durchschnittswerte für LCOM sehr gering, was vermuten ließe, so Chidamber & Kemerer, dass mindestens 50% der Klassen gebundene Methoden haben. Die Anwendung an der Entwicklungsstätte A hat ein paar Ausreißerklassen´, wie der hohe Maximumwert von 200 zeigt. An der Entwicklungsstätte B gibt es fast keine Ausreißer, was durch die unterschiedlich Verteilungen deutlich wird.

3.3. Ergebnisse der Studie

Chidamber und Kemerer behaupten, die gefundenen Maße würden die 3 Nichtimplementierungsschritte von Booch's Methode für OOD messen. Wenn man sich die untenstehende Tabelle anschaut, erkennt man, dass WMC, DIT und NOC sich auf den ersten Schritt – das Identifizieren von Klassen (und Objekten) – beziehen. In diesem Schritt werden Schlüsselabstraktionen im Problemfeld identifiziert und potentielle Klassen und Objekte ausgewiesen. Die genannten Maße würden dies insofern messen, als dass WMC einen Aspekt der Komplexität der Klasse ist und DIT und NOC sich direkt auf das Layout der Klassenhierarchie beziehen. WMC und RFC würden erfassen, wie sich Objekte einer Klasse verhalten würden, wenn sie Nachrichten erhalten. Zum Beispiel würden Klassen, die große Werte von WMC und RFC haben, vielen mögliche „Responses" haben (da die Anzahl der ausführbaren Methoden potentiell groß ist). Das LCOM-Maß beziehe sich auf das „Zusammenschnüren" von Daten und Methoden innerhalb einer Klasse und sei ein Maß für die (fehlende) Bindung in einer Klasse. Daher bezögen sich WMC, RFC und LCOM auf den zweiten Schritt (die Semantiken der Klassen). In diesem Schritt wird die Bedeutung der Klassen und Objekte, die im vorigen Schritt identifiziert wurden, festgelegt, dies beinhaltet die Definition der Lebenszyklen der Objekte von der Erzeugung bis zur Zerstörung.

Die Maße RFC und CBO würden außerdem das Ausmaß an Kommunikation zwischen den Klassen erfassen und wären somit ein Maß für den dritten Schritt (die Beziehungen zwischen den Klassen), bei dem Klassen – und Objektinteraktionen, etwa das Vererbungsmuster unter den Klassen oder die Sichtbarkeit unter Objekten und Klassen, identifiziert werden.

3.4. analytische Ergebnisse

Das Nichtzutreffen von Eigenschaft 6 bei allen Maße wird von Chidamber & Kemerer als eher positiv denn negativ empfunden. Es würde bedeuten, dass das Aufteilen einer Klasse in mehrere Klassen eher zur Vergrößerung der Komplexität denn zur Verringerung derselben führt. Dies würde durch die Erfahrungen der Entwickler gestützt. Ein Maß, das diese Eigenschaft erfülle, könnte nicht auf einer Intervall-Skala oder einer Rationalskala sein und somit keine Aussagen wie „Klasse A ist doppelt so komplex wie Klasse B." liefern.

Schlussfolgerungen:
Die Forschungen haben einen neuen Satz Maße für OOD hervorgebracht und implementiert, der sowohl theoretische Fundiertheit besitzt als auch die Ansichten von erfahrenen Softwareentwicklern darstellt. Fortsetzende Forschung sei nötig um Differenzen des objektorientierten Ansatzes und des traditionellen Ansatzes im Entwurf weiter zu untersuchen. Das Anwenden der Maße an zwei unterschiedlichen Softwareentwicklungsstätten zeige nicht nur die Möglichkeit zur Datenextraktion sondern auch Wege, wie Manager diese Maße benutzen könnten. Des weiteren würden, zusätzlich zu den normalen Vorteilen von gültigen Messens, objektorientierte Entwurfsmaße Einblicke gewähren, ob die Entwickler objektorientierten Prinzipien folgen. Die Maße könnten auch Entwicklungsleitern Gebiete der Architektur aufzeigen, die für ein Redesign in Frage kommen oder aber intensiveres Testen erfordern. Ein anderer Vorteil, den man durch die Anwendung dieser Maße bekommen würde, seien Einblicke in die Kompromisse, die von den Entwerfern gemacht werden müssen. Weil es potentiell sehr viele Entwürfe gibt, seien die Maße auch dazu geeignet, verschiedene Entwürfe gegeneinander abzuwägen. Chidamber und Kemerer räumen jedoch auch ein, dass weitergehende Forschungen Zusätze, Änderungen oder gar Entfernen von Maßen in dem vorgestellten Satz mit sich bringen könnten. Besonders das LCOM-Maß könne verschiedene Interpretationen zulassen, da es derzeit auf der datenbasierten Bindung beruhe. Es ist aber auch möglich, den Satz an Maßen als verallgemeinerte Lösung zu verwenden, um spezielle Maße, etwa für bestimmte Zwecke oder Umgebungen, zu entwickeln. [1]

4. aufbauende empirische Studien

4.1. Sharble & Cohen 1993

Die zwei Boing - Ingenieure Sharble und Cohen verwendeten die 6 Maße von Chidamber und Kemerer, um die bessere von zwei objektorientierten Entwurfsmethoden zu ermitteln. Dies ergab sich aus der Tatsache, das die zwei Ingenieure zwei unterschiedliche Entwurfsmethoden beherrschten. Es stellte sich heraus, dass die so genannte „Responsibility-based" Methode, bei der man sich auf das Modellieren des Verhaltens der Objekte konzentriert gegenüber der „Data-driven" Methode, bei der man sich darauf konzentriert, die Daten und die Zustände der Objekte zu modellieren, im Vorteil war. [45]

4.2. Li & Henry 1993

Die Wissenschaftler Li und Henry untersuchten, unter Verwendung von 5 der 6 Maße von C.& K. und weiteren 5 Maßen, Wartungs- und Änderungsdaten zweier verschiedener Systeme, und das über einen Zeitraum von 3 Jahren.

Die Studie zeigte bei beiden Systemen sehr niedrige Werte für NOC und DIT. Außerdem war eine Korrelation von WMC und RFC mit dem Umfang zu beobachten. [48]

4.3. Basili, Briand, Melo – 1995

Das Paper zeigt die Resultate einer Studie, die an der Universität Maryland durchgeführt wurde, in der experimentell der Satz Maße von CK untersucht wurde. Dazu wurden die Maße als 'Vorhersager' von defektanfälligen Klassen verwendet. Die Studie ist komplementär zu Li & Henry 1993, wo dieselben Maße verwendet wurden, um Häufigkeit der Wartungsveränderungen von Klassen zu überprüfen. Um die Validation ordnungsgemäß durchzuführen, wurden Daten der Entwicklung von 8 mittelgroßen Information - Management - Systemen, denen dieselben Anforderungen zugrunde lagen, gesammelt. Alle 8 Projekte wurden mit Hilfe von sequentiellen life cycle Modellen, einer wohlbekannten Analyse/Entwurfsmethode und der C++ Programmiersprache entwickelt. Anhand der experimentellen Ergebnisse wurden die Vorteile und Rückschläge dieser OO Maße diskutiert.
Ergebnisse: Fünf der sechs Maße sind geeignet, um die Fehleranfälligkeit vorherzusagen. Eine Ausnahme stellt hierbei das LCOM – Maß dar. Dies rührt aus der Definition von LCOM, dass LCOM „0“ ist, wenn die Anzahl von Klassenpaaren mit gemeinsamen Variableninstanziierungen größer als die ohne ist. Diese Definition ist auf keinen Fall angemessen, da sie die Binding auf ‚0’ setzt bei Klassen mit sehr unterschiedlicher Bindung. Dadurch wurden die Forscher davon abgehalten, den tatsächlichen Einfluss der Bindung basierend auf ihren Daten zu ermitteln.
Es wurde zudem festgestellt, dass die meisten dieser Maße Indikatoren zu sein scheinen, die relativ unabhängig von einander sind. Weiterhin wurde gezeigt, dass CK-Maße bessere Vorhersager als die 'besten' Mengen von 'traditionellen' Maßen zu sein scheinen. [9]

4.4. Cartwright & Shepperd 1996

Cartwright und Shepperd beschreiben das Extrahieren verschiedener Maße (unter anderen die CK-Maße DIT und NOC) aus dem Subsystem eines großen Telekommunikationssystems (133,000 loc of C++). Die Hauptentdeckung war eine Korrelation zwischen dem DIT-Maß von CK und der Anzahl der Benutzerberichte über Probleme. Im übrigen kamen Zweifel an der Effektivität des Gebrauchs der Vererbung auf, da auch hier von relativ wenig Gebrauch in dem analysierten System berichtet wurde. Erschwerend kam hinzu, dass Klassen, die die größte Änderungsdichte aufwiesen, relativ weit oben in der Vererbungshierarchie waren. [8]

4.5. Chidamber, Darcy, Kemerer 1998

In der zweiten groß angelegten empirischen Studie mit Chidamber und Kemerer wurden drei verschiedene Financial - Service - Anwendungen untersucht. Gemessen wurden alle CK-Maße, außerdem Produktivität (Umfang/Aufwand), Wiederverwendungsaufwand (Entwicklungsumfang, gemessen in Stunden) und Entwurfsaufwand (Entwicklungsaufwand, gemessen in Stunden). Die Werte für DIT und NOC waren durchweg sehr niedrig, vermutlich aus den bereits genannten Gründen, z.B. aufgrund der größeren Verständlichkeit bei geringerer Vererbungstiefe. Weiterhin waren WMC, CBO und RFC stark korreliert. Außerdem wurden hohe Werte von CBO und LCOM mit geringerer Produktivität, höherem Wiederverwendungsaufwand und höherem Entwurfsaufwand verbunden. Durch eine niedrige Varianz von DIT und NOC und der starken Korrelation von WMC, CBO und RFC, kombiniert mit einer schrittweisen Regressionsanalyse, kamen Chidamber, Darcy und Kemerer zu dem Schluss, dass die einzig signifikanten Maße für die zu messenden Faktoren (trotz unterschiedlicher Programmiersprachen und der Untersuchung von immer nur einem Faktor pro System) in dieser Studie CBO und LCOM waren. Dies, so Chidamber & Kemerer, hebe die Wichtigkeit der Konzepte von Kopplung und Bindung im Entwurf hervor. [2]

4.6. Masuda, Sakamoto Ushijima -1999

In dieser Studie wurde die Effektivität des Anwendens von Design Patterns in OO Systemen quantitativ mittels Maßen von C. & K. untersucht. Es wurden von einer Forschungsgruppe (nicht die Forscher selber) zwei verschiedene Anwendungen für die „Knowledge Discovory in Databases“ (KKD) entwickelt. Die Maße wurden von zwei verschiedenen Versionen der Anwendungen extrahiert: Das „Prototype Release“, das ohne die Nutzung von design patterns entworfen wurde und Release 2, das mit Nutzung von design patterns entworfen und implementiert wurde.
Schlussfolgerungen: Bestimmte design patterns tendieren dazu, spezifische Messwerte schlechter zu machen. Wie auch immer, das Verschlechtern von Messwerten in einigen design patterns sei auf die Charakteristik dieser design patterns zurückzuführen. Das bedeutet, dass sie nicht notwendigerweise nachteilig für den objektorientierten Entwurf seien. Statt dessen ließe dies vermuten, dass einige CK Maße nicht angemessen scheinen, um die Qualität von Entwürfen mit design patterns zu messen. Das heißt, es müssten dafür neue Maße geschaffen werden. Design patterns beschreiben die Beziehung der Klassen untereinander. In dem Satz an Maßen von C. & K. sind WMC, DIT, NOC und LCOM essentiell für einzelne Klassen. Sie seien daher zwangsläufig nicht geeignet, um die Benutzung von Design Patterns zu messen. [45]

4.7. Harrison & Counsell 2000

In diese Studie wurde die empirische Evaluation der Ebene der Vererbung in fünf objektorientierten Systemen beschrieben. Die studierten Systeme variieren sowohl in Größe und Anwendungsgebiet. Resultate der Studie mit anderen kürzlich durchgeführten Studien schienen die These zu untermauern, dass Vererbung entweder nur spärlich oder falsch benutzt wird. Die statistische Korrelation zwischen vier Vererbungsmaßen und einer Menge von abhängigen Faktoren (non-comment source lines, software understandibility, known errors und error density) liefern Beweismaterial bezüglich dieser Behauptung. Es ist außerdem nicht klar, dass Systeme, die Vererbung benutzen, notwendigerweise wartbarer sind als solche, die dies nicht tun. Die analysierten Daten von zwei der untersuchten Systeme lassen vermuten, dass tiefere Vererbungsbäume Eigenschaften eines Systems sind, die schwerer zu verstehen und (implizit) schwerer wartbar sind. Es wird analysiert, warum das der Fall ist und es werden Wege vorgeschlagen, diese Situation zu beheben. Untersucht wurden fünf C++ - Systeme mit den zwei CK-Maßen zur Vererbung und zwei weiteren vererbungsbasierten Maßen. In dieser Studie zeigten gefundenen Fehler in drei der Systeme keinerlei Beziehung zu irgendeinem der Vererbungsmaße. Vererbungsorientierte Maße, die in den 5 Systemen zum Einsatz kamen, zeigten außerdem einen Mangel an Beziehungen zu drei der abhängigen Faktoren (number of non-comment source code lines, the number of known errors und Fehlerdichte). Wie auch immer, es wurde eine interessante Beziehung zwischen dem Vererbungsbaum und der Software-Verständlichkeit gefunden.
Implikationen der Resultate:
In OO-Systemen kann die Vererbung als eine Form der Kopplung angesehen werden, derart, dass um die Funktionalität einer Klasse zu verstehen, man auch andere Klassen verstehen muss. Folglich würden Klassen auf dem 'Boden' der Vererbungshierarchie schwerer wartbar sein als solche, die weiter oben liegen. Vom Standpunkt der Wartung würde dies verzeihen, dass man entweder keine oder relativ geringfügige Vererbung in einem System oder Vererbung in Form kleiner, disjunkter Unterbäume hat. Die Gestalt der Wälder von Vererbungsbäumen kann auch das Ausmaß der Code- Wiederverwendung in objektorientierten Systeme beeinflussen. Die Resultate von System eins und zwei zeigen, dass die Verständlichkeit (und implizit die Wartbarkeit) kleiner wird, wenn DIT größer wird. Ein möglicher Grund für die Verwirrung bezüglich der Vererbung ist, dass Vererbung selbst kein leicht zu erlernendes Konzept ist und auch nicht einfach zu benutzen ist.. Die Benutzung von C++ - Friends als eine einfache Alternative zur Vererbung könnte auch den Mangel an Vererbung in C++ Systemen erklären. Interessanterweise zeigte das NOC-Maß keinerlei signifikante Korrelationen in irgendeiner der Tabellen. Eine mögliche Erklärung dafür ist, das

Warter und Entwickler Vererbung nicht auf der "Breite"-Basis sehen (nebeneinander auf der selben Ebene). Das DIT-Maß liefert eine Tiefensicht, die hinsichtlich Wartbarkeit relevanter ist, wenn man betrachtet, welche Klassen beeinflußt werden, wenn man Modifikationen durchführt. Letztens, konträr zu dem was ursprünglich gedacht wurde, könnte es sein, dass die meisten OO-Systeme einfach nicht zugänglich für Benutzung von Vererbung sind und ihre Funktionalität geht nicht mit der Benutzung von Vererbung konform.

Ergebnisse:
1. Die Natur des Anwendungsgebietes hat großen Einfluß auf die resultierende Architektur des Systems. Klare Muster tauchen in Gestalt von Vererbungshierarchien und der Funktionalität, die eine Klasse liefert, auf.
2. Die Motivation, Vererbung zu benutzen und die Faktoren hinter der Benutzung werden immer noch nicht gut verstanden. (große Anzahl von 'Friend-Konstrukten' in zwei der Systeme - möglicherweise um Vererbung aus dem Wege zu gehen). [47]

4.8. Bierman, Yang - 2001

In dieser Fallstudie wurden 39 Versionen von einem sich entwickelnden, industriellen Softwaresystem analysiert um zu sehen, ob es eine Beziehung zischen „design patterns“, anderen Entwurfsmerkmalen und der Anzahl von Veränderungen gibt. Es wurde ein starker Zusammenhang zwischen der Klassengröße und der Anzahl der Veränderungen gefunden. Die Studie, unter Verwendung von, unter anderem DIT und NOC von Chidamber & Kemerer, wurde an einem mittelgroßen kommerziellem OO-System, implementiert in C++, durchgeführt. Erfahrene OO-Entwickler, die unter anderem Gebrauch von OO design patterns und einem Versionskontrollsystem machten. Der Schwerpunkt der Studie lag auf einer spezifischen Transformation – der Transformation von Version A(erste stabile Version) und Version B (finale Version). Hierbei interessierte man sich für die Klassen, die in beiden Versionen erschienen (191).
Die Wissenschaftler stellten drei Hypothesen auf:
1. Größere Klassen sind anfälliger für Veränderungen. Eine größere Klasse hat mehr Funktionalität. Deswegen ist die Wahrscheinlichkeit größer, dass ein Teil der Funktionalität angepasst oder erweitert werden muss.
2. Klassen, die an design patterns teilhaben sind weniger anfällig für Veränderungen. Patterns sind so entworfen, dass Veränderungen eher durch Unterklassen oder durch Hinzufügen von 'Teilhaberklassen' denn durch Modifizierung bereits vorhandener Klassen geschehen.
3. Klassen, die öfter durch Vererbung wiederverwendet werden sind weniger anfällig für Veränderungen. Das heißt Klassen mit mehr direkten Unterklassen werden weniger oft verändert werden. Die Erwartungen gehen dahin, dass Nachfahren öfter hinzugefügt und modifiziert werden als Elternklassen. Dies resultiert aus der Schwierigkeit beim Modifizieren von Klassen mit vielen Nachfahren und Unterklassen - jede Veränderung einer Oberklasse beeinflusst potentiell einen Nachfahren.

Die Studie bestätigte die Hypothesen 1. und 3. jedoch war, was Hypothese 2. betrifft, sogar das Gegenteil der Fall - Die Fallstudie zeigt nicht, dass design patterns Anpassungsfähigkeit unterstützen. Eine informelle Analyse lässt vermuten, dass Klassen, die an design patterns teilhaben Schlüsselfunktionalität liefern, was erklären könnte warum diese Klassen relativ häufig verändert werden. [5]

5. Analysen

5.1. Reißing 2001

Reißing diskutiert in seinem Papier den Widerspruch zwischen der anerkannten Verbesserung der Entwurfsqualität durch das Benutzen von Design Patterns und der Verschlechterung der Entwurfsqualität durch das Benutzen von Design Patterns beim Messen. Zunächst adressiert er den unklaren Begriff Qualität. Wenn Qualität nur als Flexibilität angesehen wird, sei der Entwurf mit Design Patterns natürlich besser. Wenn man Qualität hingegen lediglich als Umfang und Kopplung auffasst, ist freilich die Variante ohne Design Patterns die bessere. Die zweite Frage, so Reißing, sei die die danach, was überhaupt gemessen wird. Wenn man Design Patterns favorisieren wollte, könnte man ein Maß konstruieren, wie etwa „Anzahl der Design Pattern Uses" um Qualität zu messen, leider seien die Dinge aber nicht so einfach. Design Pattern könnten nämlich auch missbraucht werden, wenn man sie nicht mit Bedacht einsetzt. Es sei ohnehin ein häufiges Problem von Anfängern, die dächten, dass das Benutzen von Design Patterns selbst gut ist. Deshalb sei dieses Maß, für sich betrachtet, ein schlechter Qualitätsindikator. Reißing schlägt daher vor, in die Definition von Qualität beides einfließen zu lassen – Komplexitätskriterien, wie etwa Umfang oder Kopplung, und Flexibilitätsaspekte, wozu auch Design Patterns gehören. [29]

5.2. Abounader & Lamb

In dieser Studie vergleichen die Wissenschaftler traditionelle und objektorientierte Maße, geben eine Liste mit den meisten bekannten (1997) Sätzen an Maßen an und zeigen einige der verschiedenen Richtlinien, mit denen objektorientierte Maße klassifiziert werden. Es wird darauf hingewiesen, dass es keine Übereinkunft bezüglich eines deutlichen Unterschiedes zwischen traditionellen und objektorientierten Maßen unter den Forschern gibt. Gegner der Benutzung von traditionellen Maßen innerhalb der Objektorientierung argumentieren traditionelle Maße könnten nicht Konzepte wie Vererbung oder Polymorphismus erfassen, weil sie ursprünglich für prozedurale Methoden und Sprachen entwickelt wurden. Weiterhin wird argumentiert, dass der traditionelle Ansatz mehr Aufwand für das Kodieren und Warten benötigt und dass bei objektorientierten Methoden die Betonung eher in den frühen Phasen wie Analyse und Design liegt – somit impliziere dies den Bedarf an neuen Maßen, um diesen Unterschied zu erfassen. Am Beispiel das Maßes LOC seien weiterhin deutlich die Schwächen der traditionellen Maße, wenn man sie in objektorientierten Systemen einsetzt, zu erkennen, weil nur ein kleiner Teil der Methoden eines Objektes für dieses einzigartig seien, was dem Konzept der vererbungsbasierten Wiederverwendung geschuldet wäre. Auf der Gegenseite wird damit argumentiert, dass traditionelle Maße bereits gut definiert, getestet und von Forschern und Programmierern gut verstanden sind. Sie geben jedoch zu, dass man zusätzliche Maße definieren muss, um alle objektorientierten Aspekte zu erfassen. Trotzdem gibt es Kritik dahingehend, dass viele Forscher dazu tendieren würden „das Rad neu zu erfinden" d.h. es gibt eine Tendenz unnötigerweise wohlbekannte Maße, lediglich in objektorientierten Begriffen, erneut zu definieren. Besonders Chidamber und Kemerer werden aufgezählt, sie würden z.B. LCOM für objektorientierte Systeme erneut definiert haben, obwohl gezeigt wurde, dass die Bindung einer Klasse mit Begriffen der traditionellen Bindung erklärt werden kann. Abounder und Lamb interessieren sich nicht besonders für diese Diskussion weil sie ein Datenmodell mit vielen verschiedenen Maßen aufstellen wollen, womit man sowohl traditionelle als auch objektorientierte Maße von der selben Information nehmen kann. Somit könne man sie unvoreingenommen miteinander vergleichen. [23]

5.3. Letha Etzkorn 1997

In dieser weiterführenden Analyse untersuchte die Wissenschaftlerin verschiedene Definitionen und Implementationen des LCOM-Maßes (Berücksichtigung der Vererbung oder nicht, Berücksichtigung von Konstruktoren oder nicht). Es wurde herausgefunden, dass das von Chidamber & Kemerer definierte LCOM-Maß mehrere eklatante Schwachpunkte hat:

1. Klassen mit sehr unterschiedlicher Bindung können die gleichen LCOM-Werte bekommen (wenn die Anzahl der Paare mit gemeinsamen Instanzvariablen gleich bzw. größer der Anzahl der Paare von Methoden ohne gemeinsame Instanzvariable ist).
2. Klassen mit gleicher Bindung können unterschiedliche LCOM-Werte erhalten.
3. Die Größe des LCOM-Maßes ist auf $\frac{n}{2}$ begrenzt, wobei n die Anzahl der member functions der Klasse ist. Dies kann dazu führen, dass bestimmten nichtgebundenen Klassen riesige Werte für LCOM zugeordnet werden, die sogar viel größer als „normale" LCOM-Werte sind. [34]

5.4. Rosenberg / NASA

In ihrer Arbeit „Applying Object Oriented Metrics" empfielt sie, die Maße von Chidamber & Kemerer zusammen mit einigen traditionellen Maßen für die Risikoanalyse zu benutzen. Sie definiert aufgrund des großen Erfahrungsschatzes der NASA mit objektorientierter Technologie Grenzwerte für die Maße um Risikoklassen zu identifizieren und zwar wie folgt:
Jede Klasse, die mindestens zwei der folgenden Kriterien erfüllt, wird als kritisch gekennzeichnet:
RFC > 100;
CBO > 5;
RFC > 5 * WMC mit k=1 für alle Methoden;
WMC > 100 WMC mit k=1 für alle Methoden > 40
Diese Werte sind aufgrund der sehr spezifischen Anwendung bei der NASA nicht ohne weiteres auf andere Projekte übertragbar – es soll hier vielmehr verdeutlicht werden, wie ein Vorgehen im Umgang mit Entwurfsmaßen innerhalb einer Softwarefirma grob aussehen kann. [31]

6 Bilanz

Wenn man nun eine Bilanz ziehen soll, inwieweit die Maße von Chidamber und Kemerer nützlich sind, um Erkenntnisse über die Software und den Softwareentwicklungsprozess zu bekommen, findet man einige positive Seiten, aber auch viele Negativaspekte. Wie schon vorher festgestellt, kann man die Maße dazu benutzen, um Einblicke in die Vorgehensweisen und die Kompromisse, die Entwickler im Entwurf machen müssen, zu bekommen. Des weiteren können „Risikoklassen" identifiziert werden, denen in der Testphase mehr Ressourcen zukommen müssen. So kann man auf einer höheren Ebene auch verschiedene Entwürfe auf bestimmte Qualitätsmerkmale hin gegeneinander abwiegen. Die in den bisherigen Studien betrachteten Qualitätsfaktoren, die im Allgemeinen mit hohen Werten von CK-Maßen einhergingen, waren z.B. niedrige Produktivität, hoher Aufwand Klassen wiederzuverwenden, hoher Aufwand Klassen zu entwerfen, Schwierigkeiten Klassen zu implementieren, die Anzahl von wartungsbedingten Veränderungen, die Anzahl der fehlerhaften Klassen und Problemen, die von Benutzern berichtet werden – also mehrheitlich schlechte Qualitätseigenschaften. Dies ist einfach darin begründet, dass es sich bei den CK-Maßen um Komplexitätsmaße handelt. Es kann und muss jedoch erhebliche Kritik geäußert werden. Die mangelnde Wohldefiniertheit des LCOM-Maße wurde bereits erwähnt, zudem sind die CK-Maße generell zu allgemein und müssen auch immer an verschiedene Programmiersprachen angepasst werden, denn Maße sind zwar programmiersprachenunabhängig zu extrahieren, aber bezüglich ihrer Interpretation sind sie es eben nicht. Also was nützt es einem, etwas überall messen zu können, wenn man ohnehin unterschiedlich interpretieren muss? Um Kopplung oder Bindung zu messen sind CK-Maße viel zu ungenau daher gibt es auch Forschungen, etwa verschiedene Kopplungsarten zu unterscheiden [6, 12, 13, 14]. Ferner stehen die Messwerte der so genannten Entwurfsmaße zur Entwurfszeit teilweise noch gar nicht zur Verfügung. Am Beispiel von CBO und LCOM wird

dies deutlich, wenn man sich vor Augen hält, dass es eine Implementationsentscheidung ist, auf welchen Instanzvariablen eine Methode operiert. Es ist zudem ein allgemeines Problem beim Messen, dass man abstrakte Begriffe wie Kopplung oder Bindung nur indirekt messen kann – im Beispiel das LCOM-Maßes über gemeinsame genutzte Variablen.

7 Ausblick

In der Bilanz wird bereits deutlich, dass man noch nicht verstanden hat, wie Entwurfsmaße funktionieren – man hat wohl noch nicht einmal das Zusammenwirken von objektorientierten Konzepten verstanden und Begriffe wie Kopplung oder Bindung sind doch noch abstrakt. Daher ist es erforderlich, weitere empirische Studien durchzuführen, um mehr Daten zu sammeln um irgendwann einmal vielleicht eine einheitliche Modellvorstellung zu erhalten. Es gibt bereits Bestrebungen die zahlreichen Entwurfsmaße „unter einen Hut“ zu bekommen. Des weiteren wurde am Anfang bereits gesagt, dass sich mit hoher Wahrscheinlichkeit nicht alle Anforderungen an ein Maß erfüllen lassen werden. Daher wird man für jede zu messende Eigenschaft das jeweils beste Maß zu finden haben, klar ist hierbei, das die CK-Maße viel zu allgemein gehalten sind. Das Konzept des Polymorphismus wurde bisher so gut wie nicht erfasst (zumindest auf Klassenebene). Dies ist ein Indiz dafür, wie kompliziert dieser Mechanismus ist. Wie auch immer, keines der 6 CK-Maße erfasst Polymorphismus – ein Grund für das Hinzufügen eines entsprechenden Maßes zu gegebener Zeit. Es gibt Bestrebungen, mit Hilfe bestimmter Visualisierungen objektorientierte Konzepte für den Menschen verständlicher zu machen [33] – weitere Forschungen in dieser Richtung sind lohnenswert. Letztendlich gilt es also weiterhin Daten in verschiedenen Anwendungsbereichen, Programmiersprachen und für unterschiedliche Plattformen zu sammeln – man wird Maße nur mit Hilfe von sehr zahlreichen Studien verstehen können.

References

[1] – Shyam R. Chidamber, Chris F. Kemerer – „A Metrics Suite for Object Oriented Design”
[2] – Shyam R. Chidamber, David P. Darcy, Chris F. Kemerer – “Managerial Use of Metrics for Object-Oriented Software: An Exploratory Analysis”
[4] – E. Weyuker – “Evaluating software complexity measures”
[5] – James M. Bieman, Dolly Jain, Helen J. Yang – “OO Design Patterns, Design Structure, and Program Changes: An Industrial Case Study”
[6] – Lionel Briand, Prem Devanbu – “An Investigation into Coupling Measure for C++”
[8] – Micelle Cartwright, Martin Shepperd – “An empirical View of Inheritance”
[9] – Victor R. Basili, Lionel Briand, Walcelio Melo – “A Validation of Objekt-Oriented Design as Quality Indicators”
[10] – Norman E. Fenton – “Software Metrics: A rigorous approach”
[11] - H. Zuse, “Properties of software measures,” Software Quality J., vol. 1
[12] – Lionel C. Briand, Prem Devanbu, Walcelio L. Melo – “Defining and Validating Design Coupling Measures in Object-Oriented Systems”
[13] - Lionel C. Briand, Jürgen Wüst, John W. Daly, D. Victor Porter – “Exploring the Relationships between Design Measures and Software Quality in Object-Oriented Systems”
[14] - Lionel C. Briand, Jürgen Wüst, Hakim Lounis – „Using Coupling Measurement for Impact Analysis in Object-Oriented Systems
[23] - ‚Joe Raymond Abounader, David Alex Lamb – “A Data Model for Object-Oriented Design Metrics”
[29] - Ralf Reißing - “The Impact of Pattern Use on Design Quality”
[30] – Ralf Reißing - “Towards a Model for Object-Oriented Design Measurement”
[31] - Dr. Linda H. Rosenberg - “Software Quality Metrics for Object-Oriented Environments”
[32] - Ralph D. Neal - “The Applicability of Proposed Object Oriented Metrics to Developer Feedback in Time to Impact Development”
[33] - Neville Churcher Warwick Irwin Ron Kriz – “Visualising Class Cohesion with Virtual Worlds”
[34] - Letha Etzkorn, Carl Davis, and Wei Li – “A Statistical Comparison of Various Definitions of the LCOM Metric”
[39] - I. Vessey and R. Weber, “Research on structured programming: An empiricist’s evaluation,” IEEE Trans. Sofware Eng., vol. SE-IO, pp. 394-407, 1984.
[40] - E. Weyuker, “Evaluating software complexity measures,” IEEE Trans. Software Eng., vol. 14, pp. 1357-1365, 1988.
[41] - Wand, Y. and Weber, R., Toward A Theory Of The Deep Structure Of Information Systems, International Conference on Information Systems, 1990, Copenhagen, Denmark, pp. 61-71.
[42] - Bunge, M., Treatise on Basic Philosophy : Ontology I : The Furniture of the World, Boston: Riedel, 1977.
[43] - Bunge, M., Treatise on Basic Philosophy : Ontology II : The World of Systems, Boston: Riedel, 1979.
[43] - Wand, Y. and Weber, R., An Ontological Evaluation of Systems Analysis and Design Methods, in Falkenberg, E.D. and Lindgreen, P. (ed.), Information Systems Concepts: An In-depth Analysis, Amsterdam: Elsevier Science Publishers, 1989.
[44] - Khaled El-Emam, Saida Benlarbi, Nishith Goel, Walcello Melo, Hakim Lounis, and Shesh N. Rai, The Optimal Class Size for Object-Oriented Software: A Replicated Study, March 2000
[45] - R. Sharble & S. Cohen, The Object-Oriented Brewery: A Comparison of Two ObjectOriented Development Methods , Software Engineering Notes, Volume 18, No 2, 1993, pp 60-73.
[46] - Gou Masuda, Norihiro Sakamoto and Kazuo Ushijima. Evaluation and Analysis of Applying Design Patterns. In: Procs. of the International Workshop on the Principles of Software Evolution (IWPSE99), Japan 1999.
[47] - Rachel Harrison, Steve Counsell, and Reuben Nithi. Experimental assessment of the effect of inheritance on the maintainability of object-oriented systems. In Proc. 3rd Intl. Conf. on Empirical Assessment and Evaluation in Software Engineering, University of Keele, England, 1999.
[48] - Li, Wei and Sallie Henry, "Object Oriented Metrics Which Predict Maintainability," To appear in a special issue on the object oriented paradigm, Journal of Systems and Software, November 1993.