3.1 Software Metriken

Tom DeMarco stellt in [DEMA82] fest: "Was Du nicht messen kannst, kannst Du nicht kontrollieren". Da Software ein technisches Produkt ist, muß das Produkt Software, seine Qualität und der Software Entwicklungsprozeß ebenso quantifizierbar und meßbar sein wie jedes andere industriell hergestellte Produkt. Innovative Unternehmen, wie z. B. HP, Siemens, Hitachi, Ashton Tate u. a. haben längst erkannt, daß, und wie man Produktionsprozesse, Kosten, Termine und das Produkt Software planbar und vorhersehbar gestalten kann; durch den Einsatz von Software Metriken.

Was ist Messen in der Softwaretechnik?

Ein Maß ist eine Größe zum Messen einer Eigenschaft. Die Definition von Qualitätsanforderungen geschieht durch Qualitätseigenschaften. Um Qualitätseigenschaften zu messen, müssen diese in weitere Eigenschaften bzw. Merkmale verfeinert werden. Die Merkmale werden durch Maße quantifiziert. Maße werden auf einer Skala abgebildet

3.1.1 Einige Grundlagen des Messens[1]

Eine Meßfunktion M ist eine Abbildung aus der Menge Q aller Beobachtungen eines Attributs in die Menge N der reellen Zahlen:

$$M : Q \rightarrow N$$

dabei gilt: Durch die Meßfunktion werden die empirischen Bedingungen unter den Attributmerkmalen durch die Beziehungen unter den Zahlen wiedergegeben (Repräsentationsbedingung).

Die numerischen Werte können folgende Skalen besitzen:

Skala	Erlaubte Transformationen; Vergleichsmöglichkeiten der Werte
Nominal	Permutation: w'=f(w); gleich
Ordinal	Monoton steigende Funktion: w'=f(w); gleich, größer
Intervall	Lineare Funktion: w'=αw + ß; gleich, größer (für Intervalle)
Verhältnis	Ähnlichkeitstransformation: w'=αw; gleich, größer (für Intervall u. Quotient)

Ein Sonderfall ist die absolute Skala, bei der keine Transformation erlaubt ist.

Beispiele:

Maß	Skala
Programmiersprache	nominal, Werte: C++, Modula, PASCAL usw.
F-Modularität	ordinal, Werte: Grad der Zerlegung in Prozeduren (hoch, mittel, niedrig)
McCabe	Intervall, Werte: Anzahl Entscheidungen +1
Code-Umfang	Verhältnis: Anzahl Codeeinheiten

[1] aus: M. Shepperd: Foundations of Measurement (Prentice Hall 1995)

Bei jedem Maß sollten die folgenden Angaben gemacht werden:

Name des Maßes:
Gemessenes Objekt:
Beobachtetes Attribut:
Regel für die Funktion M:
Skala:

Je präziser das Attribut formuliert ist, das gemessen werden soll, um so nützlicher sind die erhaltenen Meßwerte.

Als Beispiel die beiden Beschreibungen des McCabe-Maßes:

Maß:	McCabe
Gemessenes Objekt:	Source Code
Attribut:	Komplexität
Regel:	Anzahl Entscheidungen +1
Skala:	Intervall

Komplexität von Software ist ein recht vielseitiges und wenig präzise definiertes Attribut, so daß wir nur eine Intervall-Skala, keine Verhältnis-Skala und schon gar nicht eine absolute Skala erhalten, also ein recht mageres Ergebnis.

Wir können dagegen eine absolute Skala erreichen mit folgender Festlegung:

Maß:	McCabe-II
Gemessenes Objekt:	Source Code
Attribut:	Verzweigungen des Kontrollflusses
Regel:	Anzahl Entscheidungen
Skala:	absolut

3.1.2 Geschichtliche Entwicklung

In den 70er Jahren verbreiteten sich die strukturierten Programmiertechniken. Parallel dazu entwickelten Forscher Maße zur Erfassung der *Komplexität von Programmen* bzw. Modulen. Das Ziel war die Vermessung von Quellcode um eine quantitative Aussage über die Komplexität von Programmen zu machen. Komplexitätsmaße halfen beim Abschätzen des Aufwandes zum Testen der Programme. Man versuchte außerdem den Aufwand zum Verstehen von Programmen und der Wartungsphase daraus abzuleiten. (Halstead, McCabe).

Ende der 70er und Anfang der 80er Jahre erkannte man die Problematik dieses Vorgehens. Source Code Maße sind nur auf das fertige Produkt anwendbar. Sie stehen erst zu einem sehr späten Zeitpunkt im Projektverlauf zur Verfügung. Sie sind ungeeignet zur Abschätzung des zur Implementierung eines Softwaresystems notwendigen Aufwands. Daraus entstanden Techniken zur *Aufwand- und Kostenschätzung* von Softwareprojekten (Boehm, Albrecht).

In den 80er setzte sich die Erkenntnis durch, daß die Qualität des Entwicklungsprozesses die Qualität des Produktes bestimmt. Diese Einsicht verlagerte die Forschungschwerpunkte in die früheren Phasen der Software Entwicklung. Es entstanden:

- Maße für das Design (fan-in, fan-out in modularen Strukturen nach Henry und Kafura)
- Maße für Spezifikationen. Das klassische Maß sind die function points über die der notwendige Aufwand zur Implementierung einer Spezifikation abgeschätzt wird.
- Prozeß Maße und Modelle (Qualitätsmodelle) Das Interesse richtete sich auf das Messen von Produktivität und die Entwicklung von Kostenvorhersagemodellen. (COCOMO Modell von Boehm,

Putnams SLIM Modell).

In den 90er Jahren liegen die Schwerpunkte der Forschungs- und Entwicklungsarbeit auf ganzheitlichen Ansätzen der *Software Qualitätssicherung und Zertifizierung* von Entwicklungsprozessen (Qualitätssicherungsstandards, ISO 9000, SEI). Es entstand der neue Begriff *Software Engineering Maße*. Man erkannte, daß starre Metriksysteme bzw. Modelle nicht den Anforderungen der Praxis entsprachen und entwickelte Vorgehensweisen und Normen, die die Firmen anleiteten aus den eigenen Unternehmenszielen konkrete Metriken zur Messung der Produkte abzuleiten (GQM Ansatz, Goal/Question/Metrics von Basili, Rombach). Die ISO 9000 - Reihe ist eine Norm zur Überprüfung von Qualitätsmanagementsystemen. Die ISO 9000, Teil 3 enthält ein Qualitätssystem speziell für die Software Entwicklung.
Das Modell des Software Engineering Institut (SEI) ermöglicht die Beurteilung des Reifegrades des Enwicklungsprozesses auf einer fünfstufigen Skala (vgl. YOURD93, HUMP89).

3.1.3 Forschungsaktivitäten

Es existieren Forschungseinrichtungen, die sich mit Software Qualitätssicherung und Metriken beschäftigen. Als Beispiel seien hier die Arbeitskreise "Software-Metriken" und "Testen, Analysieren und Verifizieren von Software" der *Gesellschaft für Informatik* genannt. Die Ergebnisse der Arbeiten dieser Gruppen werden in den Software Technik Trends veröffentlicht.
Im Rahmen des *ESPRIT* Programms, das von der Europäischen Union gefördert wird, arbeiten Hochschulen, Firmen und Forschungseinrichtungen in Forschungsprojekten zu verschiedenen Themen des Software Engineering zusammen. Das Ziel ist die Entwicklung einer wissenschaftlichen Basis für das Software Engineering und der Transfer der Ergebnisse in industrielle Projekte. Nachfolgend werden einige ESPRIT Projekte aufgezählt, die sich mit Software Mertiken beschäftigen:

REQUEST - Zuverlässigkeit und Qualität von europäischer Software
MUSE - Software Qualität und Zuverlässigkeitsmetriken für spezielle Gebiete
SMART - System Vermessungs- und Architekturtechniken
MERMAID - Messen und Ressourcen Modellierungshilfen
METKIT - Metrics Education Toolkit
PYRAMID - Förderung von Metriken
MUSIC - Metriken für Benutzbarkeit Standards
AMI - Anwendbarkeit von Metriken in der Industrie

3.1.4 Frühe Metriken

3.1.4.1 Halstead-Metriken

Anzahl der unterschiedlichen Operatoren:	n_1
Anzahl der unterschiedlichen Operanden:	n_2
Anzahl des Auftretens von Operatoren:	N_1
Anzahl des Auftretens von Operanden:	N_2

Primitives Beispiel zur Veranschaulichung der Maße:

```
PROCEDURE Tausch(VAR x,y:REAL);
VAR z:REAL;
BEGIN
  z:=x;
  x:=y;
  y:=z;
END Tausch;
```

Operatoren	Anz. Auftreten	Operanden	Anz. Auftreten
PROCEDURE	1	x	3
( )	1	y	3
VAR	2	z	3
REAL	2		
:	2		
:=	3		
BEGIN..END	1		
;	6		
$n_1 = 8$	$N_1 = 18$	$n_2 = 3$	$N_2 = 9$

Abgeleitete Maße:

Größe des Vokabulars: $n = n_1 + n_2$

Länge der Implementierung: $N = N_1 + N_2$

Definition:
Berechnete Länge: $N^{\wedge} = n_1*\log(n_1) + n_2*\log(n_2)$ (1)

Es gilt empirisch: N^ ist etwa gleich N

Im Beispiel:

n = 11
N = 27
N^ = 8*log(8) + 3*log(3)
N^ = 24 + 4.75
N^ = 28.75

also etwa N^ = N.

Definition:
Programmgröße (Volumen): $V = N*\log(n)$ (2)

Diese Definition berücksichtigt folgende Überlegung:
Die n unterschiedlichen Worte können in log(n) Bits kodiert werden. Das gesamte Programm mit insgesamt N Operatoren und Operanden kann also mit V Bits dargestellt werden.
Hinweis:In Sprachen mit reichhaltigem Operatorenangebot, die dem Problem angepaßt sind, wird V klein sein, andernfalls ist V groß. Die minimale Größe für die Programmgröße V wird mit V^* bezeichnet.
Sie tritt auf, falls die Funktionen, die ein Programm bietet, voll als Befehle in der verwendeten Programmiersprache vorhanden sind.

In diesem Fall braucht i.a. kein Operand und kein Operator wiederholt zu werden.

$$V^* = (n_1^* + n_2^*) * \log(n_1^* + n_2^*) \quad (3)$$

In der Idealsprache braucht nur noch die gewünschte Funktion aufgerufen und der erhaltene Wert zugewiesen zu werden. Also existieren in der "Idealsprache" für eine Problemlösung nur noch 2 verschiedene Operatoren
Es ergibt sich im Idealfall also:

$$V^* = (2 + n_2^*) * \log(2 + n_2^*) \quad (4)$$

Im Beispiel ergäbe sich:

$V^* = (2 + 3) * \log(5) = (etwa)\ 11.6$

$V = 27 * \log(11) = 27 * 3.5 = 94.5$

Definition:
Der Level L ist definiert als:

$$L = V^* / V \quad (5)$$

Der Level hängt von der Programmiersprache ab. Im Idealfall ist der Level L = 1, real ist L < 1.

Im Beispiel gilt:

$L = 11.6 / 94.5 = 0.123$

Ein Maß für die Schwierigkeit der Implementierung D (difficulty) ist definiert als:

$D = 1/L$ (6)

Im Beispiel ergibt sich:

$D = 8.13$

Der Aufwand E (effort) ist definiert als:

$$E = V / L = V * D \quad (7)$$

Im Beispiel ergibt sich der Wert:

$E = 94.5 * 8.13 = 768.3$

Einsetzen von (5) in (7) ergibt das sehr aussagekräftige Resultat:

$$E = V^2 / V^* \quad (8)$$

Der Aufwand zur Codierung eines Programms steigt quadratisch mit der Programmgröße.

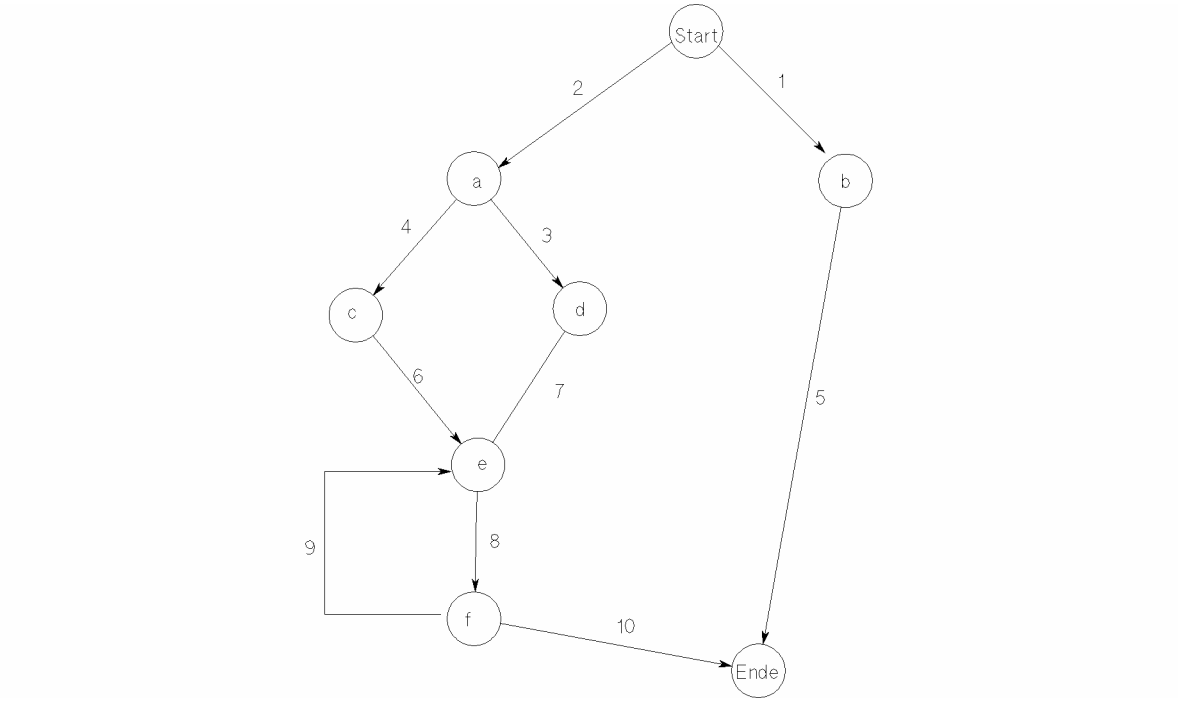
3.1.4.2 McCabe-Maß

Die zyklomatische Zahl z, die Zahl der linearunabhängigen Pfade durch den Programm-Graphen G, ist das McCabe-Maß des Programms. Für z gilt:

$z(G) = e - n + 2$

wobei gilt:
e = Anzahl der Kanten des Graphen G
n = Anzahl der Knoten von G

Beispiel:



Hier gilt: z(G) = 10 - 8 + 2 = 4
d.h es existieren 4 linearunabhängige Pfade durch den Programmgraphen G.
Die Pfade:
1) < 2,4,6,8,10 > 2) < 1,5 >
3) < 2,3,7,8,10 > 4) < 2,4,6,8,9,8,10 >

Es gilt z.B.:
< 2,4,6,8,9,8,10 > - < 2,4,6,8,10 > + < 2,3,7,8,10 > = < 2,3,7,8,9,8,10 >,
d.h. der Pfad <2,3,7,8,9,8,10> ist linear abhängig von den Pfaden unter Nr: 1), 4) und 3).
Allgemein gilt sogar, daß alle Pfade, die die Schleife <8,9> mehrfach enthalten, sind durch ein entsprechendes Vielfaches von Pfad 4) - Pfad 1) plus dem Pfad vom Anfang über e) zum Ende darstellbar, also ist z.B.:

< 2,4,6,8,(9,8),(9,8),(9,8),10 > = Pfad 1) - 3*(Pfad 4)- Pfad 1))
Die einfachste graphische Herleitung der McCabe -Zahl:
Lege eine zusätzliche Verbindung vom Ende zum Start, dann ist

z(G) = Anzahl der linear unabhängigen Zyklen des Graphen

Die McCabe-Zahl gibt Auskunft über die Komplexität der Kontrollstruktur. Falls z(G) sehr groß ist, z.B. >>10, wird der Pfad-vollständige Test sehr aufwendig.
Es gibt für einfache Programm-Graphen ein vereinfachtes Verfahren, um die McCabe-Zahl zu berechnen. Es gilt nämlich:

$$z(G) = \pi + 1$$

wobei π die Anzahl der Verzweigungen über BOOLEsche Bedingungen ist.
Im Beispiel ist gleich 3 also ergibt sich aus obiger Formel ebenso der Wert z(G) = 4.

Eine allgemein akzeptierte Regel in Bezug auf das Maß von McCabe lautet:

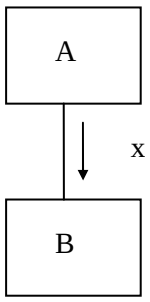
Große Programme sollten in Module zerlegt werden, wobei für jeden einzelnen Modul gelten muß, daß z(G) < 10 ist.

3.1.4.3 Maße für den Informationsfluß

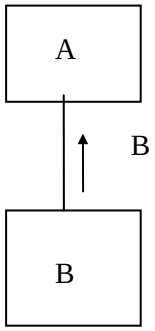
Darstellung des Flusses: (Quelle, Fluß, Ziel)

Lokaler Datenfluß (A, x, B)

Eingabeparameter x ist Werteparameter

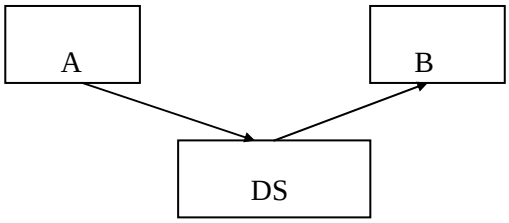


A ruft die Funktion B auf und erhält ein Ergebnis (B, B, A)



Globaler Datenfluß: (A, DS, B)

A verändert eine globale Datenstruktur und B benutzt diese



Beurteilung der Software-Architektur durch Informationsfluß-Analyse:

“fan-in“ Zahl der Datenflüsse, die zu einem Modul m führen: FI_m

“fan-out“ Zahl der Datenflüsse, die von einem Modul wegführen: FO_m

n

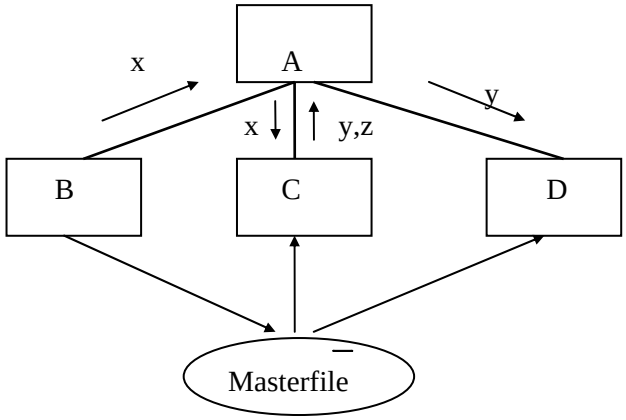
$IF4 = \sum_{i=1} (Fi_i * Fo_i)^2$ n ist Anzahl der Module

$F4_i = (Fi_i * FO_i)^2$

IF4 (S) ist eine Maßzahl für die Architektur des Systems S

$IF4_m$ (S) ist eine Maßzahl für den Modul m im System S

Beispiel:



Globale Flüsse:

(B, MF, C)
(B, MF, D)

Modul	fan-in	fan-out	FI * FO	$IF4_i$
A	3	2	6	36
B	0	3	0	0
C	2	2	4	16
D	2	0	0	0

IF4 drückt die Kopplung der Module mit ihrer Umwelt aus:

IF4(D) = IF4(B) < IF4(C) <IF4(A)

Dies bedeutet in diesem Falle, daß A eine sehr starke Kopplung mit der Umgebung hat.

IF4 kann benutzt werden, um Architekturen miteinander zu vergleichen. Deutlich geringeres IF4 bedeutet weniger starke Kopplung unter den Komponenten.

Beim "Altern" der Software entartet sehr oft die Struktur. IF4 und die verschiedenen $IF4_i$ und IF4 (S) können aufzeigen, ob und wo ein Re-Engineering notwendig, bzw. sinnvoll wäre.

Empirische Studien haben gezeigt, daß eine starke Abhängigkeit zwischen den obigen

Informationsfluß-Maßen und dem Wartungsaufwand besteht.
Es gibt aber auch Abweichungen von dieser empirischen Gesetzmäßigkeit, z.B. verursacht durch ungewöhnlich große Moduln.

Eine wichtige Frage: Welches ist die optimale Größe von Moduln?

Zu diesem Zweck können kombinierte Maße wie (LOC_i , $IF4_i$) oder ($McCabe_i$,$IF4_i$)
benutzt werden, um ausgeartete Module zu lokalisieren.

3.1.5 Metriken heute

Diese frühen Arbeiten führten zu vielen weiteren Forschungsaktivitäten. Die Akzeptanz von Metriken vergrößerte sich und führte schließlich zur Zertifizierung von gesamten Entwicklungsprozessen. Die Schwerpunkte liegen heute aber nicht mehr auf starren Metriksystemen. Moderne Metriksysteme sind flexibel und an die speziellen Bedürfnisse der Firmen anpaßbar. Sie stellen Bewertungssysteme für die Ausprägung von Qualität dar.

Was ist Software Qualität

Qualität ist nach DIN 55 350 Teil 11 definiert als die Gesamtheit aller Eigenschaften und Merkmale eines Produktes oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse beziehen. Zur Definition der Eigenschaften von Software werden Software Qualitätseigenschaften in Begriffshierarchien strukturiert. Auf der untersten Hierarchieebene werden Qualitätseigenschaften in quantitativ meßbaren Merkmalen beschrieben. Diese Begriffshierarchien werden als Qualitätsmodelle bezeichnet. Der Qualitätsbegriff wird dadurch operationalisiert. (vgl. [FRICK95], [WALL90]).

Klassifizierung von Maßen

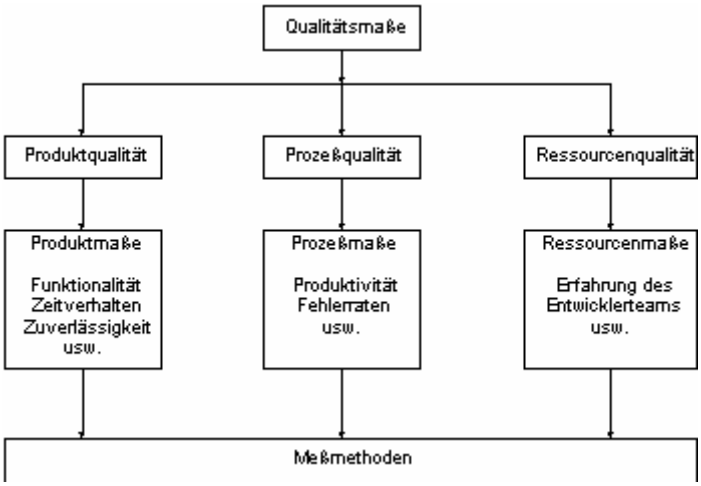


Abbildung 0.1: Klassifizierung von Metriken

Abbildung 0.1stellt eine abstrakte Übersicht zur Klassifizierung von Metriken dar. Es ist ein Ausschnitt einer Grafik des ESPRIT-METKIT-Projektes. Möller, Paulish beschreiben in [MÖPA93] ein ähnlich strukturiertes Modell.

Qualitätsmaße lassen sich nach dem Zweck ihres Einsatzes in:

- Produktorientierte
- Prozeßorientierte und
- Ressourcenorientierte Maße gliedern.

Die prozeßorientierten Maße beziehen sich auf die Qualität des Entwicklungsprozesses der Software. Der Software Entwicklungsprozeß ist das Meßobjekt. Prozessorientierte Maße definieren die Reife des Enwicklungsprozesses (z. B. Produktivität, Aufwand). Produktorientierte Metriken definieren die Qualität des Produktes, also der Software selbst (z. B. Funktionalität, Zuverlässigkeit, Portabilität). Unter den ressourcenorientierten Metriken werden die Anforderungen an das Entwicklungspersonal und die notwendige Hardware/Software definiert (z. B. Erfahrung der Entwickler). Die Hauptressource in der Software Entwicklung ist das Entwicklungspersonal, denn es verursacht die größten Kosten.

Rombach benutzt in [ROMB87] die folgenden drei Klassifikationen von Maßen:

- primitiv/komplex
- direkt/indirekt
- subjektiv/objektive Maße.

Primitive Maße sind direkt meßbare Größen (Anzahl Module, LOC-Lines of Code). Komplexe Maße kombinieren mehrere primitive Maße (LOC/Modul).
Direkte Maße sind Maße, die von keinem anderen Maß abhängen. Sie können direkt am Objekt gemessen werden (z. B. LOC). Indirekte Maße werden aus direkten Maßen abgeleitet. Sie werden daraus berechnet und können nicht selbst gemessen werden (z. B. die Produktivität berechnet sich aus LOC dividiert durch den dafür notwendigen Aufwand).
Objektive Maße sind eindeutig an einem Objekt zu messen (z. B. entdeckte Fehleranzahl).
Subjektive Maße werden benutzt um schwer quantifizierbare Sachverhalte zu beschreiben (z. B. Kundenzufriedenheit). Die Meßdaten werden auf einer subjektiven Skala abgebildet (z. B. hohe, mittlere, schwache Ausprägung eines Merkmals). (vgl. [ROMB87]).

Beispiel eines Qualitätsmodells

In diesem Abschnitt wird das Qualitätsmodell des Testtools *Logiscope* der Firma *Verilog* vorgestellt. Logiscope unterstützt dynamische und analytische Prüfungen von Software. Unter dynamischem Testen versteht man das klassische Testen durch Programmausführen. Der Source Code kann instrumentiert werden um so eine Aussage über die Vollständigkeit der durchgeführten Tests zu machen. Unter Instrumentieren versteht man die Modifikation eines Programms für den dynamischen Test, z. B. den Einbau eines Zählers in jeden Programmzweig. Am Zähler kann abgelesen werden wie oft jeder Zweig ausgeführt wurde. Bei analytischen Prüfungen (statische Analyse) wird der Source Code untersucht, vermessen und grafisch dargestellt. Er wird nicht ausgeführt. Logiscope ist für die Sprachen C, C++ und weitere Sprachen verfügbar. Die Codeanalyse erfolgt automatisch. Die zu analysierende Anwendung wird mit Hilfe der in einem Qualitätsmodell (Abb. 3) definierten Metriken geprüft. Für jedes Maß können die zulässigen Minimal- und Maximalwerte definiert werden. Die am Source Code gemessenen Werte können in verschiedenen Diagrammarten visualisiert werden. Logiscope stellt eine große Anzahl an primitiven Maßen zur Verfügung. Aus diesen primitiven Maßen können komplexe Maße für eigene Qualitätsmodelle aufgebaut werden.

Struktur des Qualitätsmodells

Logiscope enthält ein eigenes Qualitätsmodell, das aus den Modellen von Boehm und McCall abgeleitet wurde. Es kann an eigene Bedürfnisse angepaßt werden. Der Benutzer kann zusätzlich eigene Qualitätsmodelle erstellen, die nötigen Metriken definieren und diese alternativ zum Standardmodell verwenden.
Die Qualitätseigenschaften ("**factor**") werden in Merkmale ("**criteria**") zerlegt, für die dann Maße ("metrics") definiert werden. Diese drei Abstraktionsebenen definieren die Qualität aus der Sicht verschiedener Benutzergruppen von Software.

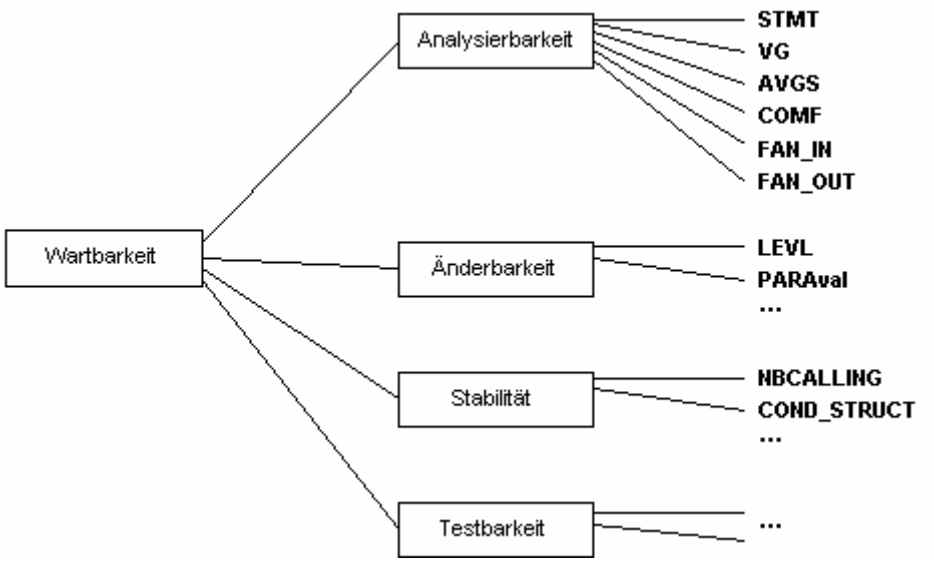


Abbildung 0.2: Ausschnitt des Qualitätsmodells des 'Tools Logiscope

- Qualitätseigenschaften ("factors") spezifizieren die Qualität aus der Sicht eines Anwenders.
- Qualitätsmerkmale ("criteria") definieren die Qualität aus Sicht des Entwicklers.
- Maße ("metrics") stellen die Sicht der Qualitätssicherung dar. Mit Hilfe der Maße kann die produzierte Software gegen das Qualitätsmodell geprüft werden.

Abbildung 0.2 stellt einen Ausschnitt der Struktur des Qualitätsmodells von Logiscope dar.

Die Maße sind *primitive* oder *komplexe* Maße; sie werden entweder am Quellcode direkt gemessen oder stellen eine Kombination mehrerer primitiver Maße dar. Die Werte komplexer Maße werden aus den gemessenen, primitiven Maßen berechnet.
Für jedes Maß wird das zulässige *Minimum* und *Maximum* definiert.
Die Ausprägung der Merkmale wird über Gleichungen aus den Maßen berechnet. Die Maße werden entsprechend ihrer Wichtigkeit für die Überprüfung des Merkmals gewichtet.
Die möglichen Wertebereiche für jedes Merkmal und für jede Metrik werden in Klassen eingeteilt (z. B. exzellente, gute, durchschnittliche, schwache Ausprägung). Diese Klassen werden für statistische Auswertungen verwendet, z. B. zur Feststellung wieviel Prozent der Funktionen weisen eine schlechte Ausprägung eines Merkmals auf und müssen nachgebessert werden.
Für jede Ausprägungsklasse eines Merkmals (bzw. einer Metrik) werden Aktionen definiert, die den Entwickler informieren, was er tun muß um eine Komponente aus einer niederen Qualitätsklasse in eine höhere zu überführen, sie also qualitativ zu verbessern. So kann z. B. für eine Komponente mit zu geringer Kommentarfrequenz die Aktion "ausführlich Kommentieren" lauten. Diese Aktionen werden dem Entwickler, bzw. der Qualitätssicherung im Qualitätsreport angezeigt.

Am Beispiel des Merkmals Analysierbarkeit sei die Struktur des Modells hier näher betrachtet. Die Analysierbarkeit charakterisiert den notwendigen Aufwand zur Diagnose von Mängeln oder Fehlern in den einzelnen Komponenten einer Anwendung. Der Aufwand hängt stark von der inneren Komplexität einer Komponente ab.

Die Maße haben folgende Bedeutung:
Über die Faktoren f1 bis f6 können die Metriken gewichtet werden, um sie auf diese Weise entsprechend ihrer Wichtigkeit für eine bestimmte Anwendung zu berücksichtigen.
Aus den berechneten Werten für die Merkmale werden die Werte für die Eigenschaften berechnet. Auch die Merkmale können dabei gewichtet werden.

STMT	Anzahl der Statements einer Komponente. Je größer die Anzahl der Statements in einem Programm, um so schwerer ist es zu verstehen, d. h. um so größer ist der Aufwand zur Fehlersuche oder bei Änderungen in der Wartung. Es ist deswegen sinnvoll die Anzahl Statements in jeder Funktion eines Programms zu beschränken.
VG	zyklomatisches Maß nach McCabe (vgl. detaillierte Darstellung in diesem Kapitel).
AVGS	durchschnittliche Länge der Statements. Dieses Maß ist ein komplexes Maß und wird nach folgender Gleichung berechnet: AVGS=(N1+N2+1) / (STMT+1) wobei N1 = Gesamtanzahl der Operatoren N2 = Gesamtanzahl der Operanden. Dieses Maß stellt die durchschnittliche Anzahl von Operatoren und Operanden für ausführbare Statements dar. Je länger eine Anweisung ist, um so größer ist der Aufwand sie zu verstehen.
COMF	Kommentarfrequenz. Das ist das Verhältnis von Kommentaren und Anweisungen.
FAN_IN	Zahl der Datenflüsse, die zu einem Modul führen. Je größer diese Zahl für einen Modul ist, um so stärker wird der Modul durch seine Umgebung (die rufenden Module) beeinflußt.
FAN_OUT	Zahl der Datenflüsse, die von einem Modul wegführen. Je größer diese Zahl für einen Modul ist, um so größer ist der Einfluß dieses Moduls auf seine Umgebung. Dieser Modul spielt also eine sehr wichtige Rolle in einem System und muß besonders genau überprüft werden.
	Aus diesen Metriken wird ein Wert für die Analysierbarkeit berechnet. Die Formel dafür lautet: Analysierbarkeit = f1*STMT + f2*VG + f3*AVGS + f4*COMF + f5*FAN_IN + f6*FAN_OUT

Darstellung der Meßergebnisse

Die Ergebnisse, die der Logiscope Static Analyzer erzeugt, können in verschiedenen Diagrammtypen dargestellt werden. Die Diagramme gewähren der Qualitätssicherung eine übersichtliche Darstellung der erzielten Qualität einer Anwendung. Die meisten vorhandenen Diagrammtypen können für die gesamte Anwendung oder für Teile davon erzeugt werden. So kann beispielsweise eine Statistik für die gesamte Anwendung, für einen Zweig eines Aufrufgraphen oder für einen einzelnen Modul erzeugt werden. Der Benutzer kann also die Granularität, auf der er die erzielten Ergebnisse betrachtet, frei wählen und die Qualität der einzelnen Komponenten einer Anwendung beurteilen.

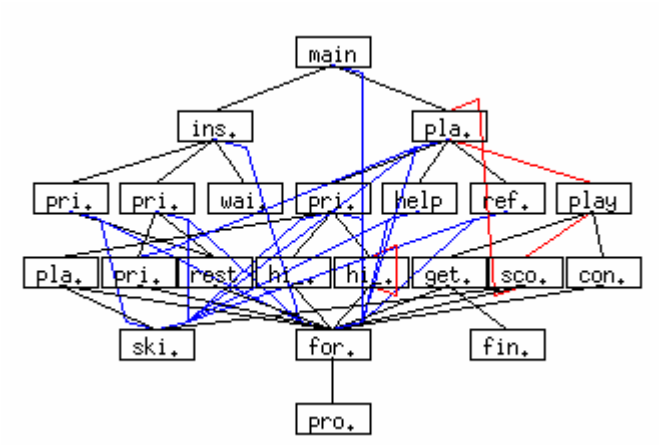
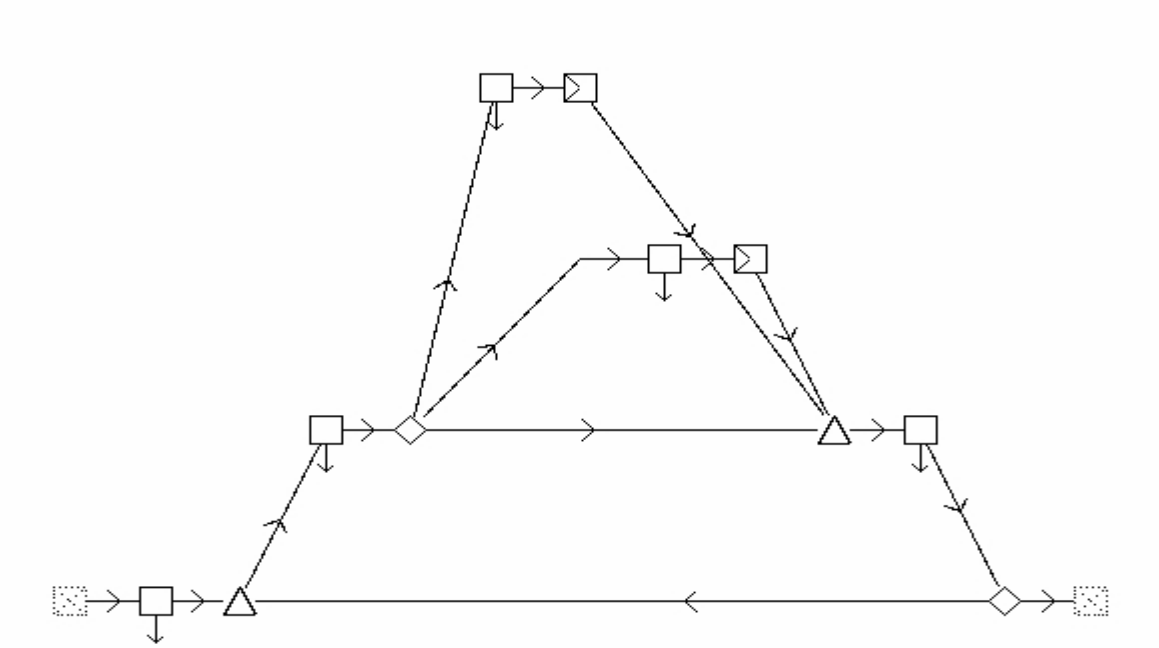


Abbildung 0.3: Aufrufgraph (call graph)

Der **Aufrufgraph** (Abbildung 0.3) zeigt die gesamte Architektur einer Anwendung. Die Knoten stellen Komponenten (Funktionen, Prozeduren) dar und die Kanten repräsentieren Aufrufe. Durch



Anklicken einer Komponente kann man in einem Editor den entsprechenden Quellcode darstellen.

Abbildung 0.4: Kontrollflußgraph (control graph)

Mit **Kontrollflußgraphen** (Abbildung 0.4) wird die Kontrollstruktur von Komponenten dargestellt. Jeder Knoten repräsentiert eine Anweisung. Kanten repräsentieren den möglichen Kontrollfluß zwischen den Knoten.

Kiviatdiagramme (**Fehler! Verweisquelle konnte nicht gefunden werden.**) erlauben die Überprüfung der gemessenen Werte auf Einhaltung der zulässigen Minima und Maxima. Jeder Radius stellt eine Metrik dar. Der innere Kreis repräsentiert das zulässige Minimum, der äußere Kreis das Maximum. Mit diesem Diagrammtyp können Maße, die außerhalb des erlaubten Bereichs liegen einfach identifiziert werden. Kiviatdiagramme lassen sich auf Komponenten oder auf gesamte Softwaresysteme anwenden. Auf der linken Seite des Diagramms werden die gemessenen und die erlaubten Werte tabellarisch dargestellt.

METRIC	LO	HI	VALUE
N_STMTS	1	50	2
PR_LGTH	3	350	9
VG	1	15	1
MAX_LVLS	1	5	1
N_PATHS	1	80	1
UNCOND_J	0	0	0
COM_R	0.20	1.00	1.50
AVG_S	3.00	7.00	4.50
VOC_F	1.00	4.00	1.28
N_IO	2	2	2

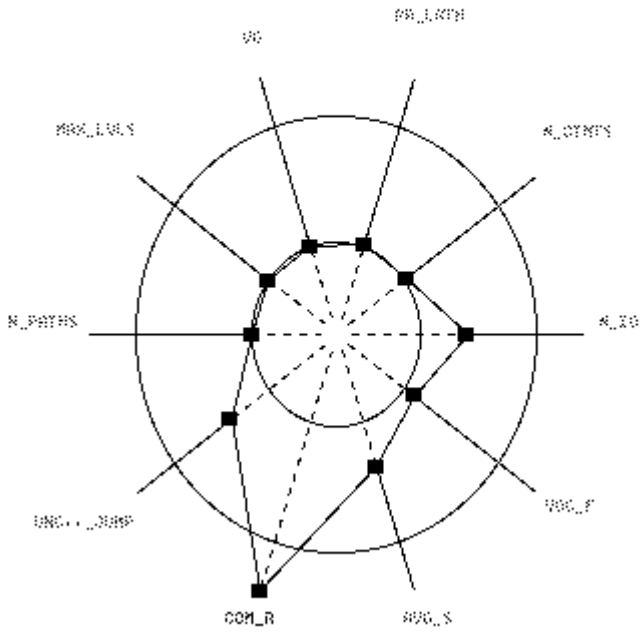


Abbildung 0.5 Kiciatdiagramm

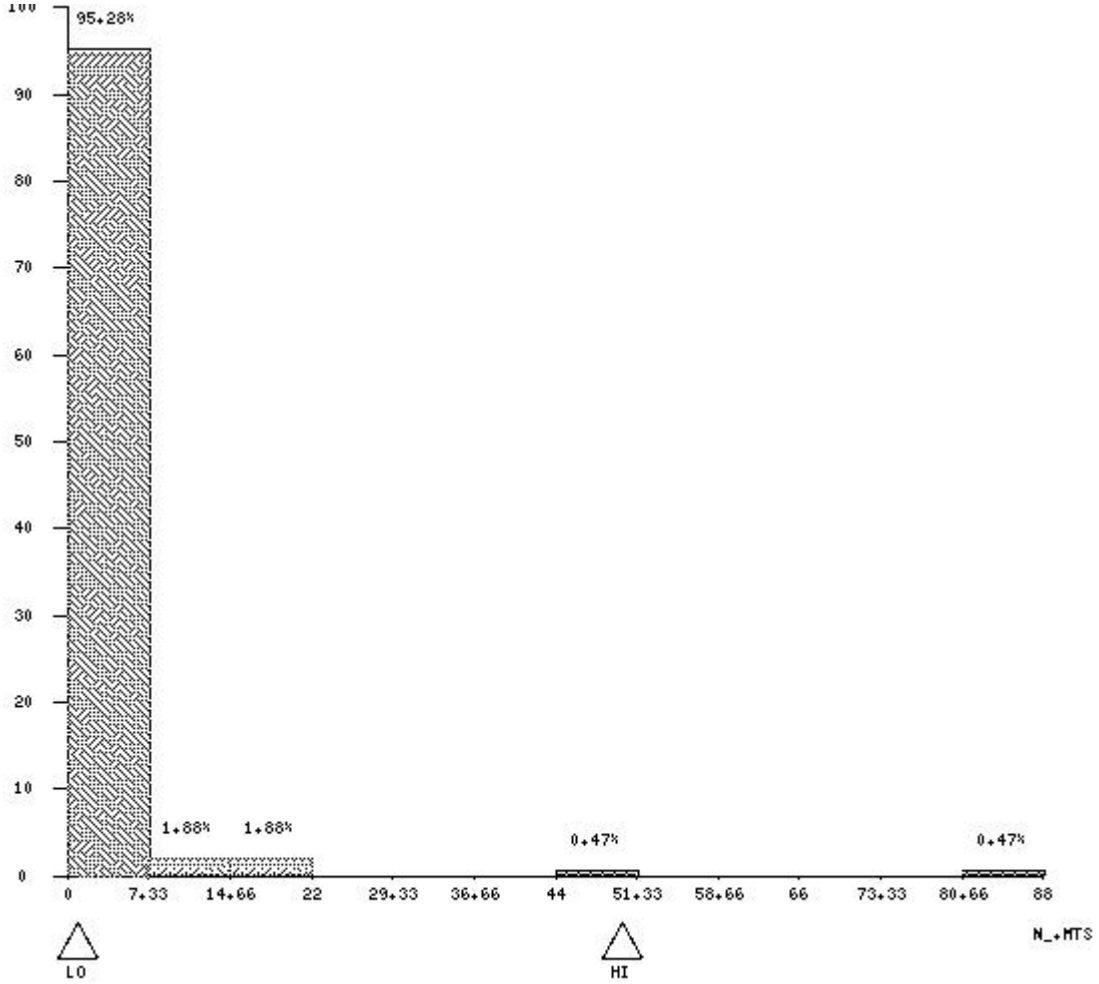


Abbildung 0.6: Statistikdiagramm

Logiscope stellt **Statistikdiagramme** zur Verfügung die statistische Auswertungen der gemessenen Werte erlauben. Sie zeigen die Verteilung der Komponenten entsprechend der gemessenen Werte der verschiedenen Metriken. Das Diagramm in Abbildung 0.6 zeigt die Verteilung der Komponenten einer Anwendung über der Anzahl Statements. 74,3 % der Komponenten liegen im erlaubten Bereich (zwischen Lo und Hi). Die übrigen Komponenten liegen außerhalb des erlaubten Bereiches.