UNIVERSITÀ DEGLI STUDI DI CAGLIARI
FACOLTÀ DI INGEGNERIA ELETTRONICA

Corso di Ingegneria del Software
Prof. Giuliano Armano

Metriche del Software

Nicola Asuni
matr. 16747

Nicola Barrocu
matr.17535

Anno Accademico 2000-2001

Indice

1. Introduzione

I costi del software sono attualmente una delle componenti principali di molti budget aziendali e il loro dimensionamento fa crescere il bisogno di metodi oggettivi e confrontabili al fine di riconoscere e dimensionare le cause che li hanno generati.

A questo scopo vengono in aiuto strumenti quali le metriche del software che nell'ambito della loro varietà e molteplicità forniscono misure caratteristiche del prodotto e del processo che lo ha generato, anche nell'ottica di migliorare la qualità e la produttività associata allo sviluppo ed al mantenimento del software.

Le metriche e il loro utilizzo rivestono quindi particolare importanza non solo dal punto di vista del produttore, ma anche da quello dell'utilizzatore finale del software.

Lo studio delle metriche del software ha avuto inizio negli anni sessanta e solo attualmente sta diventando una scienza matura. Il loro sviluppo e utilizzo, grazie anche alle risorse investite nella ricerca da parte di importanti aziende e gruppi governativi (es. NASA), ha seguito di pari passo l'evolversi negli anni del prodotto software e delle sue tecniche di ingegnerizzazione, passando dall'approccio procedurale a quello object-oriented (orientato agli oggetti).

2. Definizione di metrica

Per poter introdurre il concetto di metrica definiamo innanzitutto la **misura** come l'assegnamento empirico e oggettivo di un numero o simbolo a un'entità per caratterizzarne uno specifico attributo.

In generale le **metriche** sono degli standard di misura. Il termine *metriche* è spesso usato anche per indicare un insieme di misure specifiche definite per un particolare processo.

Nell'ingegneria del software le metriche vengono usate per quantificare e qualificare:

- *i prodotti dell'ingegneria del software,* cioè i progetti, il codice sorgente e i casi di test;
- *i processi dell'ingegneria del software* cioè le attività di analisi, progettazione e codifica;
- *gli ingegneri del software* cioè l'efficienza di un certo collaudatore, o la produttività di un certo progettista.

3. Utilità delle metriche

Le *metriche del software* giocano un ruolo chiave nell'ingegneria del software.

Per poter migliorare le prestazioni, la produttività e la qualità, le organizzazioni necessitano di poter "misurare" il software ad ogni stadio del ciclo di vita.

Il ricorso alle metriche dovrebbe facilitare lo sviluppo di modelli capaci di descrivere, ma anche di predire, attributi di processo e di prodotto.

In generale le *metriche del software* sono utili per:

- Migliorare il processo software
- Pianificare, seguire e controllare l'andamento di un progetto
- Valutare la qualità del prodotto
- Stimare lo sforzo richiesto per sviluppare un sistema software

4. Caratteristiche delle metriche

Le *metriche del software* dovrebbero essere indipendenti dallo sviluppo tecnico e dalle scelte di implementazione, così da poter essere utilizzate per confrontare la produttività di differenti tecniche e tecnologie.

Le metriche del software dovrebbero essere:

- Semplici
- Oggettive (il più possibile)
- Ottenibili ad un costo ragionevole
- Valide
- Robuste

Le metriche del software relative all'object-oriented si differenziano anche per le seguenti caratteristiche:

- Localizzazione (Localization)
- Incapsulamento (Encapsulation)
- Occultamento dell'Informazione (Information Hiding)
- Eredità (Inheritance)
- Tecniche di Astrazione (Object Abstraction Techniques)

Localizzazione (Localization)

Consiste nel collocare gli elementi in stretta vicinanza fisica l'uno con l'altro:

- I processi di decomposizione funzionale localizzano le informazioni vicino alle funzioni.
- L'approccio orientato ai dati localizza le informazioni vicino ai dati.
- L'approccio object-oriented (orientato agli oggetti) localizza le informazioni vicino agli oggetti.

In molti software convenzionali (software creati usando la decomposizione funzionale), la localizzazione è basata sulle funzionalità. La maggior parte delle metriche è quindi stata tradizionalmente legata ai concetti di funzioni e funzionalità.

Nella programmazione object-oriented, invece, la localizzazione è basata sugli oggetti. questo significa che:

- Sebbene si possa parlare delle funzionalità fornite da un oggetto, è necessario che una parte consistente delle metriche considerate possa riconoscere la natura dell'*oggetto* come componente elementare del software.
- All'interno dell'approccio object-oriented, la localizzazione fra funzioni e oggetti non sempre è una relazione uno a uno, in quanto una funzione può coinvolgere diversi oggetti, e a un oggetto possono corrispondere diverse funzioni.

Incapsulamento (Encapsulation)

È l'impacchettamento di un certo insieme di elementi.

Il meccanismo di impacchettamento può avvenire a basso livello, come nel caso di record e array, oppure a un livello superiore, come nel caso di sottoprogrammi (cioè procedure, funzioni, subroutines). Nei linguaggi di programmazione object-oriented ci sono dei meccanismi di incapsulamento a livello ancora maggiore, come le classi del C++ e i packages dell'Ada.

In molti linguaggi di programmazione object-oriented l'incapsulamento di oggetti (cioè le classi e le loro istanze) è sintatticamente e semanticamente supportato dal linguaggio stesso. In altri, il concetto di incapsulamento è supportato concettualmente ma non fisicamente.

Il più importante impatto dell'incapsulamento sulle metriche è che l'unità di base più lunga non sarà il sottoprogramma, bensì l'oggetto.

Occultamento dell'Informazione (Information Hiding)

È la soppressione (o l'occultamento) di dettagli. L'idea generale consiste nel mostrare soltanto le informazioni necessarie per realizzare lo scopo immediato.

Ci sono diversi gradi di occultamento dell'informazione che vanno da una parziale limitazione della visibilità ad una totale invisibilità.

Notare che incapsulamento e occultamento dell'informazione non sono la stessa cosa, cioè un elemento può essere incapsulato ma ancora totalmente visibile.

L'occultamento dell'informazione gioca un ruolo diretto nelle metriche per quanto riguarda l'accoppiamento degli oggetti e il grado di occultamento dell'informazione.

Eredità (Inheritance)

È un meccanismo attraverso il quale un oggetto acquisisce una o più caratteristiche da altri oggetti.

Alcuni linguaggi object-oriented supportano solo una singola eredità, cioè un oggetto può acquisire caratteristiche direttamente da un unico singolo oggetto; altri invece supportano un'eredità multipla, cioè un oggetto può acquisire caratteristiche direttamente da due o più oggetti differenti.

I tipi di oggetti che possono essere ereditati e le particolari semantiche variano da linguaggio a linguaggio.

Come vedremo in seguito, esistono molte metriche dell'ingegneria del software object-oriented basate sull'eredità, come le metriche *Number Of Children*, *Number Of Parents* e *Depth of Inheritance Tree*.

Astrazione (Object Abstraction Techniques)

È un meccanismo che serve per focalizzare l'attenzione sui dettagli più importanti ed essenziali di concetti o elementi, ignorando i dettagli meno significativi.

L'astrazione è un concetto relativo nel senso che, considerando livelli più alti di astrazione, si ignorano sempre più dettagli, cioè si fornisce un panorama più generale di un certo concetto o elemento. Analogamente, considerando livelli più bassi, si introducono più dettagli, cioè si fornisce un panorama più specifico di un certo concetto o elemento.

Esistono diversi tipi di astrazione, per esempio: astrazione di funzioni, di dati, di processi e di oggetti. In particolare nell'astrazione di oggetti, questi vengono trattati come entità di alto livello, cioè come black boxes.

5. Criteri di valutazione delle metriche

Nell'approccio tradizionale di decomposizione funzionale e di analisi dei dati, le metriche misurano in maniera indipendente la struttura dei dati e la struttura del progetto.

Le metriche object-oriented invece devono essere adatte a trattare le funzioni e i dati che si presentano integrati insieme in un oggetto.

Per valutare la capacità di una metrica di misurare la qualità di un software, bisogna basarsi sulla misura degli attributi associati alla qualità del software, quali:

- *Efficienza*: riferita alla progettazione dei costrutti.
- *Complessità*: i costrutti devono poter essere utilizzati in maniera più efficiente al decrescere della complessità dell'architettura.
- *Comprensibilità*
- *Riuso*: un progetto di qualità deve poter essere riusato
- *Collaudabilità e Manutenzione*: la struttura deve supportare facilmente il testing e i cambiamenti.

6. Tipi di metriche tradizionali

Le metriche sono sostanzialmente di due tipi:

- *Metrica Primitiva (diretta):* è una misura, è quindi direttamente osservabile (es: linee di codice, tempo totale di sviluppo, …)
- *Metrica Calcolata (indiretta):* non è direttamente osservabile ma viene calcolata a partire da altre metriche (es: numero difetti / linee di codice, …)

Possiamo classificare le metriche del software in tre grandi gruppi (metriche di Prodotto, di Processo e di Qualità) secondo lo schema seguente:

Metriche di Prodotto

Metriche Dimensionali (Size Metrics)

Le metriche dimensionali si usano per normalizzare le misure di qualità e produttività rispetto alle dimensioni del sistema software realizzato.

- Linee di Codice (LOC - Lines Of Code o SLOC – Source Lines Of Code)
- Nessun Commento – Nessuno Spazio (NCNB - Non-Comment Non-Blank o NLOC o ELOC – Effective Lines OF Code)
- Espressioni Eseguibili (EXEC - Executable Statement)
- Linee di Commento (CLOC – Comment Lines Of Code)
- Percentuale di Commenti (CP - Comment Percentage)
- Punti Funzione (FP - Function Points)
- Punti Funzione Estesi (SPR - Feature Points)

Metriche Di Complessità (Complexity Metrics)

- Complessità Ciclomatica (CC - McCabe's Cyclomatic Complexity)
- Complessità Ciclomatica Estesa (Myer’s Cyclomatic Complexity)
- Flusso di Informazione (Information Flow)

Metriche Di Halstead (Software Science Metrics)

- Vocabolario di Programma (n - Program Vocabulary)
- Lunghezza dell’Implementazione (N - Implementation Length)
- Volume di Programma (V - Program Volume)
- Lunghezza di Programma (N^ - Program Length)
- Volume Potenziale (V* - Potential Volume)
- Livello del Programma (L - Program Level)
- Difficoltà di Programma (D - Program Difficulty)
- Sforzo di Programmazione (E - Programming Effort)
- Tempo di Sviluppo (T)

Metriche di processo

Le metriche di processo offrono un punto di osservazione strategico per la valutazione dell'efficacia di un progetto software.

Metriche di Maturità

- Organizzative
- Di Risorse
- Di Personale
- Di Gestione Tecnologica

Metriche di Gestione

- Del Progetto
- Della Qualità
- Di Configurazione

Metriche di Ciclo di Vita

- Di Definizione del Problema
- Di Analisi
- Di Progetto
- Di Implementazione

Metriche di Qualità

Metriche di Affidabilità

- Probabilità di malfunzionamento in un dato intervallo temporale
- Tempo medio prima di un malfunzionamento (Mean Time to Failure)

Metriche di Manutenibilità

- Corregibilità
- Collaudabilità
- Espandibilità

7. Metriche di Prodotto

7.1. Metriche Dimensionali (Size Metrics)

7.1.1. Linee di Codice (LOC - Lines Of Code)

LOC (Lines Of Code) o SLOC (Source Lines Of Code) è la metrica dimensionale più semplice: conta tutte le linee fisiche del codice sorgente. Questa definizione di LOC non è rigorosa in quanto su una riga si possono trovare più istruzioni o parti di istruzioni.

Le soglie per valutare il significato delle misure con metriche del tipo LOC variano in dipendenza dal linguaggio e dalla complessità del metodo.

Questa metrica quindi non tiene conto della diversa potenza dei linguaggi e non fornisce una relazione tra produttività e qualità.

7.1.2. Nessun Commento – Nessuno Spazio (NCNB)

Sono indicate nei seguenti modi:

- **NCNB**: Non-Comment Non-Blank
- **NLOC**: Non Comment Lines Of Code
- **ELOC**: Effective Lines OF Code

Questa metrica calcola tutte le linee effettive del codice (linee del codice sorgente), cioè tutte le linee del codice eccetto i commenti e le linee vuote.

7.1.3. Espressioni Eseguibili (EXEC - Executable Statement)

Questa metrica conta solo le linee di comando del codice indipendentemente dalle linee fisiche.

Tra le metriche basate sul conteggio delle linee di codice, l'EXEC è quella meno dipendente dal programmatore o dallo stile di programmazione, pertanto risulta più adatta quando un programma è scritto con più linguaggi diversi.

7.1.4. Linee di Commento (CLOC – Comment Lines Of Code)

Questa metrica conta solo le linee di commento del codice.

7.1.5. Percentuale di Commenti (CP - Comment Percentage)

La metrica Percentuale di Commenti (CP - Comment Percentage) è definita come la percentuale di linee di commento rispetto a tutte le linee di codice (escluse le linee vuote):

$$\mathrm{CP} = \frac{\mathrm{CLOC}}{NLOC + CLOC}$$

Data l'utilità dei commenti, questa metrica è usata per valutare gli attributi di comprensibilità, riuso, e manutenzione. Secondo alcune fonti, il CP ideale si attesta attorno al 30%.

7.1.6. Punti Funzione (FP - Function Points)

L'Analisi dei Punti Funzione (FPA - Function Point Analysis) è una misura dimensionale introdotta da Allan Albrecht alla conferenza SHARE/GUIDE/IBM a Monterey, California nell'ottobre del 1979.

Con l'introduzione di nuovi potenti linguaggi di programmazione le metriche del tipo *costo per linea di codice* (metriche LOC) non erano più sufficienti a tener conto dei costi fissi non associati alla codifica. Mentre i requisiti, le specifiche, i documenti e molti altri elementi di costo tendevano ad avere un costo fisso, i nuovi potenti linguaggi di programmazione riducevano il numero di "unità" che dovevano essere prodotte per un dato programma o sistema ma, paradossalmente, le vecchie metriche come *costo per linea di codice* calcolavano un aumento dei costi anziché una diminuzione.

I Punti Funzione (FP – Function points) sono un tipo di misura funzionale capace di quantificare un'applicazione software da punto di vista dell'utente, ricavano un indice delle dimensioni di un programma mediante la misura indiretta delle funzionalità che deve fornire.

In sostanza i FP forniscono una misura oggettiva e comparativa che assiste nella valutazione, progettazione, amministrazione e controllo della produzione software.

Il principale problema dei FP è che banalizzano la definizione di complessità, che dipende sostanzialmente dallo I/O e non dalle caratteristiche dell'algoritmo.

Le caratteristiche principali dei FP sono:

- Misurano le funzionalità richieste e ricevute dall'utente.
- Misurano il ritmo e la dimensione dello sviluppo e del mantenimento indipendentemente dalla tecnologia usata per l'implementazione.
- Forniscono una misura normalizzante per le altre metriche o per altri progetti e organizzazioni.
- Consentono di convertire la dimensione di un'applicazione in qualsiasi linguaggio (in linee di codice) nell'applicazione equivalente scritta con un altro linguaggio.
- Consentono di misurare la produttività di un progetto scritto con più linguaggi.

Linguaggio	LOC/FP
Assembler	320
Macro Assembler	213
C	150
Algol	106
Chill	106
Cobol	106
Fortran	106
Jovial	106
Pascal	91
RPG	80
PL/I	80
Modula 2	71
Ada	71
Prolog	64
Lisp	64
Forth	64
Basic	64
Logo	53
4th Generation Database	40
Strategem	35
APL	32
Objective C	26
Smalltalk	21
Query Languages	16
Spreadsheet Languages	6

Tabella 1: Rapporto tra linee di codice e Punti Funzione a seconda del linguaggio

Metodo Standard IFPUG (International Function Point Users Group)

Il metodo IFPUG (International Function Point Users Group) è il metodo standard per la misurazione dei FP (Function Point Counting Practices Manual, Release 4.1).

L'IFPUG (www.ifpug.org) è un'organizzazione internazionale no-profit costituitasi nel 1986 con lo scopo di divulgare dati ed informazioni relativi alla *Metrica dei Punti Funzione*.

Il calcolo dei FP può essere diviso in 8 fasi:

Fase iniziale

Prima di calcolare i Punti Funzione occorre determinare i vincoli dell'applicazione che andranno tracciati dall'esperto per il calcolo dei FP con l'esperto sulle funzionalità dell'applicazione. Il risultato di questa fase sarà il diagramma dei vincoli (boundary diagram).

Questa fase è critica nel calcolo dei FP.

Fase 1 – Pianificare il conteggio dei Punti Funzione

La fase di calcolo dei FP dovrebbe essere inclusa nel piano complessivo del progetto. Così facendo i punti funzione possono essere programmati e pianificati. La prima misura dei FP dovrebbe essere sviluppata per fornire una misura utile alla stima.

I FP possono essere calcolati prima del completamento dei requisiti. I FP calcolati nelle fasi iniziali del ciclo di vita dovranno essere accuratamente documentati in modo tale da poter essere facilmente manutenzionati ed aggiornati. Naturalmente le stime iniziali sono soggette al cambiamento dei requisiti, col cambiare della portata e dimensione del progetto cambia anche lo sforzo per il completamento del progetto. L'analisi dei FP può essere completata prima di aver completato il set dei requisiti ma dovrebbe essere completata solo dopo che i requisiti sono stati finalizzati e nuovamente implementati.

Dopo il completamento del calcolo dei FP i risultati andrebbero confrontati con i calcoli precedenti per verificare ogni componente aggiunto o modificato. Ogni aggiunta al calcolo dei FP dovrebbe avere un'etichetta indicante se il componente nuovo o aggiunto è il risultato di una modifica della funzionalità o il risultato di un miglioramento nel processo di conteggio.

Fase 2 – Raccolta della documentazione

La seguente documentazione è raccomandata al fine di assistere nel completamento del calcolo dei FP prima della finalizzazione dei requisiti:

- Documentazione che l'utente percepisce come oggettiva, problemi e necessità.
- Documentazione riguardante il sistema corrente, se tale sistema (automatico o manuale) riesce.
- Documentazione su ogni obbiettivo ricercato ed obbligo per il sistema proposto.
- Descrizione e/o modello della struttura complessiva del sistema (framework).
- Qualsiasi altra documentazione già completata.

La documentazione raccomandata per il calcolo dei Fp dopo la fase di analisi e disegno è:

- Formati di schermo e maschere di dialogo
- Schema dei rapporti
- Schema delle maschere di input
- Interfacce con altri sistemi e tra sistemi
- Modelli logici e/o preliminari dei dati fisici
- Dimensione e formato dei file
- Opzioni di menù

Il calcolo dei FP completato prima del progetto andrebbe confrontato con quello calcolato dopo il completamento del progetto. Questo sarà un indicatore di quanto l'applicazione è cresciuta in rapporto ai requisiti.

La documentazione raccomandata alla conclusione o implementazione del progetto dovrebbe includere tutta la documentazione precedente ed ogni documentazione addizionale sul sistema.

Fase 3 – Determinare il valore del Fattore di Aggiustamento (VAF)

Il *Valore del Fattore di Aggiustamento* (VAF - Value Adjustment Factor) andrebbe preso in considerazione durante le fasi iniziali del calcolo dei FP. Al procedere del calcolo dei FP il VAF andrebbe aggiornato ad ogni nuova informazione disponibile. Il VAF andrebbe aggiornato al termine di ogni fase del calcolo dei FP.

Il VAF è basato su 14 Caratteristiche Generali del Sistema (GSC - General System Characteristics) che valutano le funzionalità generali dell'applicazione. Ogni caratteristica ha una descrizione associata che ne aiuta a determinare il grado di influenza.

Il *Grado di Influenza* è rappresentato da una scala numerica da 0 a 5:

Grado di influenza	Descrizione
0	ININFLUENTE
1	INCIDENZA SCARSA
2	INCIDENZA MODERATA
3	INCIDENZA MEDIA
4	INCIDENZA SIGNIFICATIVA
5	INCIDENZA ESSENZIALE

Tabella 2: Grado di Influenza per le GSC

Il *Counting Practices Manual* dell'IFPUG fornisce dei criteri dettagliati di valutazione per ognuno dei GSC. La tavola seguente mostra uno schema riassuntivo:

	GSC (Caratteristiche generali del Sistema)	**Descrizione**
1	Comunicazione dati	Quante semplificazioni per le comunicazioni ci sono al fine aiutare il trasferimento o lo scambio di informazioni con l'applicazione o il sistema?
2	Elaborazioni dati distribuite	Come sono gestiti i dati e le funzioni di elaborazione distribuite?
3	Prestazioni	Qual è il tempo di risposta o la capacità dati richiesta dall'utente?
4	Configurazione utilizzata intensamente	Quanto intensamente è utilizzata la piattaforma hardware dove l'applicazione sarà eseguita?
5	Tasso di transazioni	Quanto frequentemente vengono eseguite le transazioni? (giornalmente, settimanalmente, mensilmente…)
6	Immissione dati on-line	Quale percentuale di informazioni saranno inserite on-line?
7	Efficienza orientata all'utente finale	L'applicazione è stata progettata per un'efficienza orientata all'utente finale?
8	Aggiornamento on-line	Quanti File Logici Interni (ILF) saranno aggiornati da transazioni on-line?
9	Complessità dell'elaborazione	L'applicazione ha una logica estesa o elaborazioni matematiche?
10	Riusabilità	L'applicazione è stata sviluppata per soddisfare i le necessità di uno o più utenti?
11	Facilità di installazione	Quanto è complessa la conversione e l'installazione?
12	Facilità operativa	Quanto efficaci e/o automatici sono le procedure di avvio, back-up e recovery?
13	Installazioni multiple	L'applicazione è stata specificamente progettata, sviluppata e supportata per essere installata su più postazioni e per più organizzazioni?
14	Modificabilità	L'applicazione è stata specificamente progettata, sviluppata e supportata per essere facilmente modificata?

Tabella 3: Caratteristiche Generali del Sistema (GSC - General System Characteristics)

Il valore del VAF sarà quindi calcolato secondo la formula seguente:

$$\mathbf{VAF = 0.65 + (0.01 \cdot S_{1,14}\, F_i)}$$

Dove F_i sono le risposte numeriche da 0 a 5 date alle 14 caratteristiche generali del Sistema (GSC).

Fase 4 – Inventario delle Operazioni e dei File

Prima di classificare singolarmente i componenti è necessario completare la raccolta dati di tutte le componenti dei FP (EI, EO, EQ, ILF e EIF). È consigliabile raccogliere prima i dati sulle transazioni e poi raccogliere i dati sui file necessari per le transazioni in modo tale da assicurarsi che i file che non siano ILF o EIF non vengano inclusi.

I Punti Funzione (FP - Function Points) sono basati su 5 "funzioni" logiche divise in 3 funzioni transazionali (transactional function types) e 2 funzioni dati (data function types):

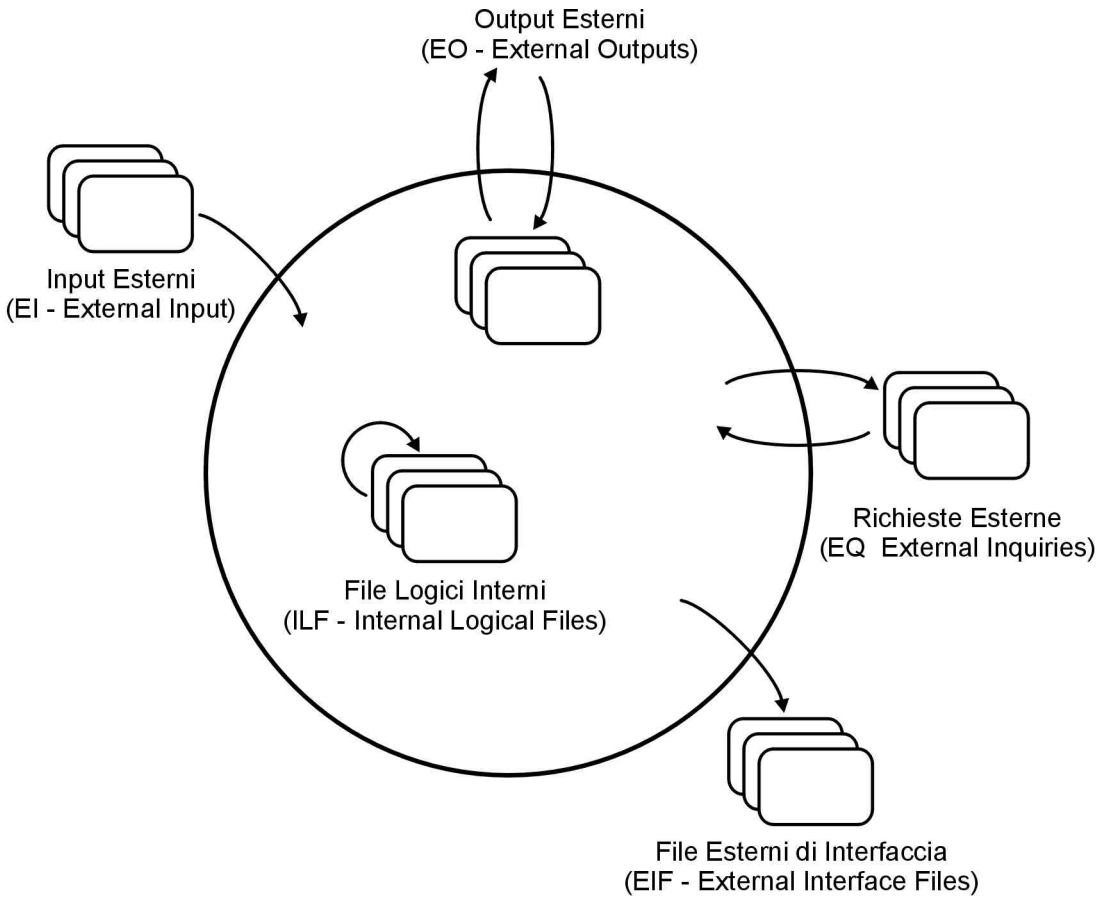


Figura 1: Funzioni Logiche del Sistema

- ***Input Esterni (EI – External Inputs):*** Processo elementare in cui i dati attraversano il limite dall'esterno all'interno. Questi dati possono arrivare da una schermata di input o da un'altra applicazione. I dati sono utilizzati per mantenere uno o più file logici. Una buona fonte di informazioni per determinare gli EI è costituita dagli schemi di schermo, formati di schermo e dialogo, schemi delle maschere di input. Gli input aggiuntivi da altre applicazioni devono essere considerati a questo punto ed aggiorneranno gli ILF delle applicazioni calcolate.
- ***Output Esterni (EO - External Outputs):*** Processo elementare in cui i dati *derivati* attraversano il limite dall'interno all' esterno. I dati creano dei rapporti o dei file di output inviati ad altre applicazioni. Questi rapporti o file sono creati da uno o più *File Logici Interni* (ILF). Una buona fonte di informazioni per determinare gli EO

sono gli schemi di rapporto e i formati di file elettronici che sono inviati all'esterno dei limiti dell'applicazione.

- ***Richieste Esterne (EQ – External Inquiries):*** Processo elementare con componenti di input ed output che risultano da un recupero da uno o più ILF. Il processo di input non aggiorna alcun ILF e la parte di output non contiene dati derivati. Una buona fonte di informazioni per determinare gli EQ è costituita dagli schemi di schermo, formati di schermo e dialogo, schemi delle maschere di input.
- ***File Logici Interni (ILF - Internal Logical Files):*** Gruppo di dati logicamente correlati identificabili dall'utente che risiedono interamente all'interno dei limiti dell'applicazione e sono mantenuti attraverso gli Input Esterni (EI). Una buona fonte di informazioni per determinare gli ILF sono i modelli preliminari di dati, schemi di tabelle, descrittori di database.
- ***File Esterni di Interfaccia (EIF - External Interface Files):*** Gruppo di dati logicamente correlati identificabili dall'utente che sono utilizzati solo per riferimento. I dati risiedono interamente all'esterno dell'applicazione e sono mantenuti da un'altra applicazione. Il File di interfaccia Esterna è un ILF per un'altra applicazione. Una buona fonte di informazioni per determinare gli EIF sono le descrizioni di interfacce con altri sistemi.

La raccolta completa di informazioni sui componenti dei FP andrebbe distribuita tra tutti i componenti del gruppo di lavoro in maniera da assicurarsi che sia stato incluso tutto. Quando si è avuta la conferma che tutte le transazioni ed i file sono stati inclusi si può procedere alla classificazione individuale dei singoli componenti.

Fase 5 – Classificazione dei componenti

È importante capire la matrice associata alle transazioni (EI, EO, EQ) ed i file (ILF, EIF). Chi esegue il calcolo deve identificare la riga appropriata prima di determinare la colonna. C'è molta meno granularità nel determinare la riga appropriata (numero di file referenziati per la transazione e numero di tipi di record per file) che la colonna appropriata. Questo aiuta a ridurre lo sforzo necessario a completare il calcolo dei FP. Estendendo questa analisi, le cose comuni andranno considerate prima del calcolo delle transazioni e file. Chi calcola i FP dovrà porsi i seguenti quesiti per accelerare il processo di calcolo:

- ***Input Esterni (EI – External Inputs):*** *Gli EI necessitano di più o di meno di 3 file per essere processati?* Per tutti gli EI che referenziano più di 3 file è necessario sapere se gli EI hanno più o meno di 4 tipi di elemento dato (DET – Data Element Types). Se gli EI hanno più di 4 DET gli EI verranno valutati con un peso *alto*, altrimenti con un peso *medio*. Gli EI che referenziano meno di 3 file saranno messi in disparte e calcolati separatamente.
- ***Output Esterni (EO - External Outputs):*** *Gli EO necessitano di più o di meno di 4 file per essere processati?* Per tutti gli EO che referenziano più di 4 file è necessario sapere se gli EO hanno più o meno di 5 tipi di elemento dato (DET – Data Element Types). Se gli EO hanno più di 5 DET gli EO verranno valutati con un peso *alto*, altrimenti con un peso *medio*. Gli EO che referenziano meno di 4 file saranno si in disparte e calcolati separatamente.
- ***Richieste Esterne (EQ – External Inquiries):*** La stessa analisi condotta per gli EI e gli EO può essere condotta per le EQ ma utilizzando il maggiore degli EI ed EO così come prescritto nel *IFPUG counting practices manual*.

- ***File Logici Interni (ILF - Internal Logical Files) e File Esterni di Interfaccia (EIF - External Interface Files):*** *Tutti i file contengono un tipo di record o più di un tipo?* Se tutti o la maggior parte dei file contiene solo un tipo di record ci occorre sapere se il file contiene più o meno di 50 tipi di elemento dato (DET – Data Element Types). Se il file contiene più di 50 DET verrà valutato con un peso *medio*, se meno di 50 con un peso *basso*. Ogni file che contiene più di un tipo di record sarà messo in disparte e calcolato separatamente.

Fase 6 – Esaminare 14 Caratteristiche Generali del Sistema (GSC)

Le 14 Caratteristiche Generali del Sistema (GSC) andranno esaminate per assicurarsi dell'accuratezza delle risposte.

A questo punto si può procedere al calcolo del VAF così come spiegato nella fase 3.

È fondamentale applicare con cura il GSC ed il VAF perché possono pesantemente influire sul calcolo dei Function Points (+/- 35%).

Fase 7 – Tabulazione dei risultati

Il calcolo di ogni livello di complessità per ogni tipo di componente può essere introdotto in una tabella come quella seguente. Ogni cifra è moltiplicata per il peso corrispondente e tutti i risultati vengono sommati nella colonna totale della riga corrispondente. La somma dei valori nella colonna totale fornisce il valore dei Punti Funzione (FP) non corretti (*Unadjusted Function Points*).

Tipo di Componente	**Complessità dei Componenti**			
	PESI			
	BASSO	MEDIO	ALTO	TOTALE
EI	…… · 3 = ……	…… · 4 = ……	…… · 6 = ……	
EO	…… · 4 = ……	…… · 5 = ……	…… · 7 = ……	
EQ	…… · 3 = ……	…… · 4 = ……	…… · 6 = ……	
ILF	…… · 7 = ……	…… · 10 = ……	…… · 15 = ……	
EIF	…… · 5 = ……	…… · 7 = ……	…… · 10 = ……	
			Totale FP non corretti	
			VAF	
			Totale FP	

Tabella 4: Calcolo dei Punti Funzione

Il totale dei Punti Funzione (FP) si ottiene moltiplicando il valore dei Punti Funzione non "Aggiustati" per il Valore del fattore di Aggiustamento (VAF).

Fase 8 – Convalida dei risultati

Il risultato del calcolo dei FP dovrà essere controllato dall'intero gruppo di lavoro e validato dal coordinatore delle metriche.

Gli errori più frequenti nel calcolo dei FP sono gli errori di omissione, altri errori sorgono quando costruzioni fisiche sono sostituite da costruzioni logiche e contate come componenti. Pertanto il gruppo di lavoro dovrà controllare l'analisi dei FP per completezza e verificare che tutte le componenti (EI, EO, EQ, ILF e EIF) siano state incluse. Il coordinatore delle metriche deve lavorare a stretto contatto con chi esegue materialmente i calcoli per assicurarsi che il processo e la validazione siano stati eseguiti correttamente.

Le seguenti sono domande che dovrà porsi chi esegue il calcolo dei FP:

1. Il conteggio dei FP è un compito del piano generale di progetto?
2. La persona che esegue il calcolo dei FP è adeguatamente istruita a farlo?
3. Chi esegue il calcolo dei FP usa la documentazione corrente per calcolare i FP?
4. Sono state seguite le linee guida dell'IFPUG?
5. Sono state seguite le linee guida interne all'organizzazione per il calcolo dei FP?
6. L'applicazione è stata calcolata dal punto di vista dell'utente?
7. L'applicazione è stata calcolata da un punto di vista logico e non fisico?
8. I vincoli stabiliti per il calcolo dei FP coincidono con i vincoli stabiliti per le altre metriche? Se no, perché?
9. Le percentuali delle singole componenti dei FP (ILF, EIF, EI, EO a EQ) rispettano le medie industriali (40% per ILF, 5% per EIF, 20% per EI, 25% per EO, 10% per EQ) e le medie stabilite per GTE? Se no, c'è una valida ragione?
10. La raccolta delle transazioni (EI, EO, EQ) e dei file (ILF, EIF) è stata controllata dal gruppo di lavoro?
11. Le risposte date ai 14 GSC ricadono nel range delle risposte date ad altri progetti nella stessa organizzazione?
12. Il calcolo è stato registrato in un archivio per i FP?
13. Le premesse sono coerenti con tutti gli altri progetti?
14. Le premesse sono state accuratamente documentate?
15. Il calcolo è stato controllato da un esperto in calcoli di FP?

Stima dei Punti Funzione

L'introduzione dei *Punti Funzione* ha rappresentato un grosso passo in avanti nella misura del software rispetto all'utilizzo tradizionale del conteggio delle *Linee di Codice*. Di contro, i Punti Funzione richiedono un livello molto dettagliato di documentazione sul software in esame. Esistono quindi almeno due situazioni in cui avere un metodo di stima compatibile ma alternativo alle regole standard dei FP può essere decisivo:

Il primo caso si verifica quando lo sviluppo o la manutenzione di un software si trova in una fase iniziale ed è semplicemente impossibile contare i FP secondo lo standard IFPUG.

Il secondo caso si verifica quando è necessaria una valutazione su un software esistente ma la documentazione o il tempo e le risorse necessarie ad eseguire un calcolo dettagliato dei FP non è disponibile.

A causa di queste ed analoghe situazioni, è cresciuta la domanda di metodi di stima (non di calcolo) da parte delle organizzazioni coinvolte nello sviluppo del software. In letteratura sono quindi presenti diversi metodi di stima dei FP tra cui citiamo: *Early Function Points, ILF Models, backfiring*.

Backfiring

Backfiring è il nome della tecnica per passare dal calcolo dei FP al numero di LOC.

Questa tecnica usa un coefficiente di trasposizione proprio dei diversi linguaggi di sviluppo (tabella 5) ed un coefficiente correttivo che tiene conto del contesto.

Coefficiente di trasposizione:

Linguaggio	Livello nominale	LOC/FP		
		minimo	media	massimo
1st Generation	1.00	220	320	500
Basic Assembly	1.00	200	320	450
Macro Assembly	1.50	130	213	300
C	2.50	60	128	170
BASIC (interpretato)	2.50	70	128	165
Fortran	3.00	75	107	160
Cobol	3.00	65	107	150
C++	6.00	30	53	125
Smalltalk	15.00	15	21	40
SQL	27.00	7	12	15

Tabella 5: Coefficienti di Backfiring

Coefficiente correttivo:

Si calcola tenendo conto di tre aspetti fondamentali ai quali si da un peso da 1 a 5:

- **Complessità del problema**
  - 1: Solo algoritmi e calcoli semplici
  - …
  - 5: Molti algoritmi e calcoli complessi

- **Complessità delle strutture dati**
  - 1: Poche variabili scarsamente correlate tra di loro
  - …
  - 5: Struttura dei file complessa con molte interrelazioni tra file

- **Complessità del codice**
  - 1: Non Procedurale
  - 2: Molto strutturato, con moduli riusabili
  - 5: Scarsamente strutturato

Il coefficiente di aggiustamento è determinato con la formula:

0.7 + 0.05 (Ct - 3)

dove Ct è la somma dei tre livelli di complessità

7.1.7. Punti Funzione Estesi (SPR - Feature Points)

I *Punti Funzione* forniscono dei dati errati in tutti quei casi in cui il software ha un'elevata complessità algoritmica ma pochi input ed output. Sia da un punto di vista psicologico che pratico questo tipo di sistemi richiedono un tipo di metodo equivalente ai Punti Funzione ma sensibile alle difficoltà causate dalla complessità algoritmica.

Per queste ragioni nel 1986 la *Software Productivity Research Inc.* ha sviluppato e sperimentato un metodo per applicare la logica dei Punti Funzione ai sistemi operativi, sistemi di commutazione telefonica e simili. Per evitare la confusione con il metodo dei Punti Funzione dell'IBM, questa alternativa sperimentale è stata chiamata *Feature Points* (SPR - Punti Funzione Estesi).

Questa metrica è stata poi applicata con successo anche al altri tipi di software: software di sistema, embedded, real time, CAD (Computer Aided Design), AI (Artificial Intelligence), software per comunicazioni, software per il controllo dei processi, simulazioni discrete, software matematico e persino MIS (Management Information Systems) per il quale erano stati originariamente inventati i Function Points.

La metrica dei Punti Funzione Estesi (SPR - Feature Points) è un *superset* della metrica dei Punti Funzione della IBM, essa infatti aggiunge ai cinque parametri standard un sesto chiamato *algoritmi* che ha un peso standard di 3. I Punti Funzione Estesi riducono il valore empirico del peso medio dei *File Logici Interni (ILF - Internal Logical Files)* da 10 a 7 in maniera da tener conto della tipologia dei sistemi analizzati.

Come si può constatare, per le applicazioni in cui il numero di algoritmi e ILF coincidono (ad es: applicazioni MIS) sia i Punti Funzione che i Punti Funzione Estesi forniscono gli stessi risultati numerici. Quando invece prevalgono in larga misura gli algoritmi sui ILF (es. software di sistema) i Punti Funzione Estesi generano un totale significativamente maggiore dei Punti Funzione, viceversa, quando prevalgono in larga misura i ILF sugli algoritmi (es: sistemi informativi) i Punti Funzione Estesi generano un totale significativamente minore dei Punti Funzione.

Tipo di componente	Complessità dei Componenti
Algoritmi	…… · 3 = ……
EI – Input Esterni	…… · 4 = ……
EO – Output Esterni	…… · 5 = ……
EQ – Richieste Esterne	…… · 4 = ……
ILF – File Logici Interni	…… · 7 = ……
EIF – File Esterni di Interfaccia	…… · 7 = ……
Totale SPR non Aggiustati	
VAF	
Totale SPR	

Tabella 6: Calcolo dei Punti Funzione Estesi (Feature Points)

A seconda della difficoltà e complessità dell'algoritmo, il metodo SPR assegna un valore al parametro *algoritmi* da 1 a 10. Al valore 1 equivalgono algoritmi con semplici operazioni aritmetiche o semplici regole, al valore 10 equivalgono algoritmi con equazioni complesse, operazioni matriciali e difficoltà di processo matematiche o logiche. Il valore per un algoritmo standard che fa uso della matematica ordinaria è 3.

Essendo la metrica SPR ancora in fase sperimentale non esiste ancora una tassonomia per la classificazione degli algoritmi oltre che metodi ad hoc basati sul tipo di impiego dell'algoritmo. In ogni caso, le basi per il calcolo del peso degli algoritmi sono due:

1. Il numero di fasi di calcolo o regole richieste dall'algoritmo.
2. Il numero di fattori o dati richiesti dall'algoritmo.

Di seguito alcune regole supplementari per valutare quali algoritmi sono calcolabili:

1. L'algoritmo deve affrontare un problema risolvibile.
2. L'algoritmo deve affrontare un problema limitato.
3. L'algoritmo deve affrontare un problema definito.
4. L'algoritmo deve essere finito e deve avere una fine.
5. L'algoritmo deve essere preciso e senza ambiguità.
6. L'algoritmo deve avere un input o un valore di partenza.
7. L'algoritmo deve avere un output o produrre dei risultati.
8. L'algoritmo deve essere implementabile, in modo tale che ogni fase possa essere eseguita da un computer.
9. L'algoritmo può includere o chiamare algoritmi subordinati.
10. L'algoritmo deve poter essere rappresentato attraverso i concetti standard della programmazione strutturata (if-then-else, do-while, case, …).

7.2. Metriche Di Complessità (Complexity Metrics)

7.2.1. Complessità Ciclomatica (CC – Cyclomatic Complexity)

La Complessità Ciclomatica (CC – McCabe's Cyclomatic Complexity) è usata per valutare la complessità di un algoritmo in un metodo ed è interamente basata sulla struttura del grafo che rappresenta l'algoritmo.

La CC è definita come il numero di casi di test necessari per testare esaurientemente il metodo.

Riferendosi ad un grafo che rappresenta l'algoritmo (vedi Figura 2) e posto:

- v(G) = numero ciclomatico relativo al grafo G
- L = numero di archi nel grafo
- N = numero di nodi del grafo
- P = numero dei componenti del grafo disconnessi.

Si ha:

$$\mathbf{v(G) = L - N + 2 \cdot P}$$

In un grafo G fortemente connesso, il numero ciclomatico v(G) è uguale al numero di percorsi linearmente indipendenti.

Per una sequenza dove è presente un solo percorso (non ci sono scelte né opzioni) allora sarà necessario un solo caso di test. Se invece è presente un *If loop* allora avrò due scelte cioè due percorsi alternativi: se la condizione è *vera* verrà testato un percorso, se la condizione è *falsa* verrà testato l'altro.

In generale se sono presenti tanti *If loop* allora avrò tante scelte che generano dei percorsi multipli, ad ognuno dei quali è associato un caso di test.

La figura 2 mostra un esempio di calcolo della CC per quattro strutture di programmi elementari.

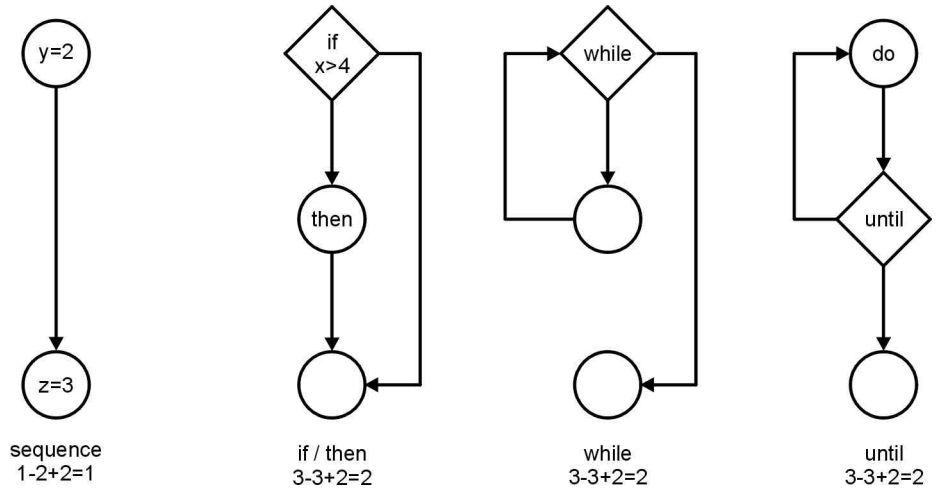


Figura 2: Esempio di calcolo della Complessità Ciclomatica (Cyclomatic Complexity)

Generalmente un metodo con CC bassa è migliore, sebbene a volte questo non è dovuto ad una bassa complessità del metodo bensì al fatto che il metodo differisce le decisioni inviando un messaggio.

A causa dell'eredità, la CC non può essere usata per misurare la complessità di una classe; tuttavia la complessità di una classe può essere valutata usando la CC di un singolo metodo combinata con altre metriche.

Così come proposto da McCabe, la CC di un metodo dovrebbe essere inferiore a 10, in quanto per CC maggiori di 10 si ha un aumento discontinuo degli errori per linea. Nella pratica sono stati utilizzati con successo CC fino a 15 anche se la gestione di questo tipo di progetti richiede una notevole preparazione.

Sebbene questa metrica sia specificatamente applicabile per valutare la complessità, essa è anche collegata a tutti gli altri attributi.

La CC ha diverse proprietà:

1. v(G) ≥1
2. v(G) = numero massimo di percorsi linearmente indipendenti in G.
3. L'inserimento o l'eliminazione di espressioni in G non influenza v(G).
4. G ha un solo percorso solo se v(G) = 1.
5. Inserendo un nuovo arco in G, v(G) aumenta di 1.
6. v(G) dipende solo dalla struttura decisionale di G.

Si può dimostrare che la CC di un programma è uguale al numero di nodi decisionali più 1:

v(G) = n° decisioni + 1

Il concetto di Complessità Ciclomatica è legato a quello di *testabilità*, infatti minore è la complessità e maggiore è la facilità di esecuzione dei test.

7.2.2. Complessità Ciclomatica Estesa (Myer's Cyclomatic Complexity)

G. Myers propose di estendere la misura della complessità tenendo conto dei seguenti aspetti:

- complessità delle operazioni booleane e le espressioni if-then nidificate.
- conteggio separato di ogni singola condizione.
- espressioni *case*.

7.2.3. Flusso di Informazione (Information Flow)

$$\mathbf{C = L \cdot (Fan\text{-}in \cdot Fan\text{-}out)^2}$$

L = metrica dimensionale (LOC)
Fan-in = numero di flussi entranti nel componente
Fan-out = numero di flussi uscenti dal componente

7.3. Metriche Di Halstead (Software Science Metrics)

Nel tentativo di studiare le proprietà misurabili degli algoritmi, Halsted ha sviluppato una metrica dimensionale per programmi scritti in vari linguaggi che utilizza un certo numero di unità elementari come unità di misura per la sintassi e la semantica. Halsted ha operato una distinzione tra i concetti di operatore ed operando, l'operando è stato definito come una variabile e l'operatore come un simbolo aritmetico.

Le metriche primitive di Halstead sono definite come:

- $\mathbf{n_1}$ = numero di operatori distinti di un programma
- $\mathbf{n_2}$ = numero di operandi distinti di un programma
- $\mathbf{N_1}$ = numero delle occorrenze di operatori di un programma
- $\mathbf{N_2}$ = numero delle occorrenze di operandi di un programma
- $\mathbf{f_1}$ = numero delle occorrenze del j-esimo operatore, j=1,2,..n_1

7.3.1. Vocabolario di Programma (n - Program Vocabulary)

Il vocabolario **n** dell'algoritmo o del programma è definito come:

$$\mathbf{n = n_1 + n_2}$$

7.3.2. Lunghezza dell'Implementazione (N - Implementation Length)

È una misura della dimensione del programma.

$$\mathbf{N = N_1 + N_2}$$

o in altri termini:

$$N_1 = \sum_{j=1}^{n1} f_{1j} \qquad N_2 = \sum_{j=1}^{n2} f_{2j} \qquad N = \sum_{i=1}^{2} \sum_{j=1}^{ni} f_{ij}$$

7.3.3. Volume di Programma (V - Program Volume)

Rappresenta la dimensione dell'implementazione e può essere pensata come il numero di bit richiesti per rappresentare l'intero programma nella sua forma minima (indipendentemente dalla lunghezza dei nomi dei token).

$$\mathbf{V = N \cdot \log_2 n = N \cdot (\log n / \log 2)}$$

7.3.4. Lunghezza di Programma (N^ - Program Length)

La Lunghezza di Programma rappresenta il numero di bit necessari a rappresentare almeno una volta tutti i token che esistono nel programma. Può essere considerato come il numero di bit necessari a rappresentare la tavola dei simboli del programma.

$$\mathbf{N^{\wedge} = n_1 \cdot \log_2 n_1 + n_2 \cdot \log_2 n_2}$$

7.3.5. Volume Potenziale (V* - Potential Volume)

È un tentativo di approssimare il numero minimo di token necessari per specificare l'algoritmo. Rappresenta il minimo numero di parametri di input ed output ed operatori necessari per la specificazione.

$$\mathbf{V^* = (2 + n_2^*) \log_2(2 + n_2^*)}$$

$\mathbf{n_2^*}$ rappresenta il numero di operandi del programma che compaiono una volta sola.

7.3.6. Livello del Programma (L - Program Level)

Rappresenta il livello dell'implementazione (è basso per linguaggi assemblativi).

$$\mathbf{L = V^*/V}$$

che può essere approssimato come:

$$\mathbf{L^{\wedge} = (2 \cdot n_2)/(n1 \cdot N2)}$$

7.3.7. Difficoltà di Programma (D - Program Difficulty)

Rappresenta la difficoltà di implementare un algoritmo in un particolare linguaggio.

$$\mathbf{D = 1/L^{\wedge} = (n1 \cdot N2)/(2 \cdot n_2)}$$

7.3.8. Sforzo di Programmazione (E - Programming Effort)

Lo *Sforzo di Programmazione* può essere pensato come lo sforzo richiesto a comprendere l'implementazione piuttosto che produrla. Si può anche pensare che E sia una misura della chiarezza del programma.

$$\mathbf{E = V \cdot D = V/L = V_2/V^*}$$

7.3.9. Tempo di Sviluppo (T)

Stima il tempo di sviluppo: $\mathbf{T = E/C}$ dove $5 \leq C \leq 20$, valore tipico C=18

8. Stima dei Costi nei Progetti Software

La stima dei costi del software fornisce il collegamento vitale tra i concetti e le tecniche generali dell'analisi economica ed il mondo particolare dell'ingegneria del software.

La stima dei costi del software è una parte essenziale della gestione del software.

I costi del software sono dovuti ai requisiti, all'hardware ed alle risorse umane. Queste ultime costituiscono il capitolo principale di costo sulle quali sono focalizzate la maggior parte delle tecniche di stima. Il costo di un progetto dipende dalla sua natura e dalle sue caratteristiche, l'accuratezza della stima dipende invece dalla quantità di informazioni attendibili che si dispongono circa il prodotto finale. Quando il progetto si trova in fase iniziale o durante lo studio di fattibilità, abbiamo solo un'idea approssimata delle funzionalità del sistema e della quantità di informazioni che verranno scambiate.

Malgrado queste limitazioni, gli attuali modelli di stima dei costi forniscono dei dati attendibili ed accurati.

La *stima dei costi del software* si focalizza su tre aspetti fondamentali:

- quanto tempo durerà lo sviluppo del sistema
- quanto lavoro richiederà in termini di anni-uomo (MY) o mesi-uomo (MM)
- quante persone vi parteciperanno

La stima dei costi può essere realizzata attraverso l'impiego di diversi modelli di previsione:

- **Modelli Algoritmici:** Questi metodi forniscono uno o più algoritmi che stimano i costi del software in funzione di un certo numero di variabili correlate con una o più metriche del software.
- **Giudizio Esperto:** Utilizza la consulenza di uno o più esperti possibilmente coadiuvati dall'aiuto di un meccanismo di consenso-esperto come la tecnica Delphi.
- **Stima per Analogia:** Mette in relazione per analogia l'attuale progetto con uno precedentemente realizzato dalle caratteristiche simili.
- **Principio di Parkinson:** Viene invocato il principio di Parkinson ("Il lavoro si espande fino ad occupare tutto lo spazio disponibile") per dimensionare il costo alle risorse disponibili.
- **Prezzo della Vittoria:** La stima di costo prodotta da questo metodo è uguale al prezzo che si crede adeguato per vincere l'appalto. La stima dello sforzo dipende dal budget del cliente e non dalle funzionalità del software.
- **Stima Top-Down:** La stima del costo è derivata dalle proprietà generali del software ed è poi divisa tra le varie componenti.
- **Stima Bottom-Up:** Si sommano le stime delle singole componenti del software.

Al fine di valutare l'utilità di un modello per la stima dei costi del software è opportuno tenere in considerazione i seguenti criteri:

- **Definizione:** Il modello ha chiaramente definito i costi che dovrà stimare e quelli da escludere?
- **Fedeltà:** La stima è vicina ai costi dei progetti precedenti?
- **Obbiettività:** Il modello evita l'assegnazione della maggior parte dei costi variabili del software per calibrare fattori soggettivi (come la compessità)? Se si, è complesso aggiustare il modello in modo da ottenere i risultati voluti?
- **Costruttore:** Può l'utente chiedere perché il modello fornisce una data stima? Questo aiuta l'utente a capire il lavoro che deve essere svolto?
- **Dettagli:** Il modello si adatta facilmente ad un sistema software costituito da un'elevato numero di sottosistemi ed unità? Fornisce informazioni sulle fasi e la suddivisione dell'attività?
- **Stabilità:** Una piccola variazione in input produce una piccola variazione nell'output della stima dei costi?
- **Portata:** Il modello è adatto alla tipologia di progetto software che si vuole stimare?
- **Facilità d'uso:** Gli input e le opzioni del modello sono semplici da capire e specificare?
- **Eventualità:** Il modello impedisce l'uso di informazioni che non siano state ben comprese fino al completamento del progetto?
- **Parsimonia:** Il modello impedisce l'uso di fattori altamente ridondanti o fattori che non danno un contributo significativo al risultato finale?

Le categorie di **Modelli di Previsione** che prenderemo in esame sono quattro:

Empirici

- Modello di Wolverton

Statistici/Empirici

- Modello di Walston e Felix

Analitici

- Modello di Putnam
- Modello di Halstead (vedi capitolo precedente)

Compositi

- COCOMO 81
- COCOMO II

8.1. Modelli Empirici

8.1.1. Wolverton

Questo modello, sviluppato da Ray Wolverton nel 1972, è basato sul *divide et impera* della decomposizione funzionale.

- Assume che il ciclo di sviluppo sia a cascata
- Usa una matrice predittiva 25×7
- Ogni modulo è classificato in base a:
  - Appartenenza ad uno tra 6 tipi (control, I/O, …)
  - Nuovo modulo (SI/NO)
  - Livello di difficoltà del progetto (basso, medio, alto)
- Viene stimata la dimensione di ogni modulo
- Il costo è valutato in confronto con progetti dalle caratteristiche simili

8.2. Modelli Statistici

I modelli statistici/empirici si basano sulla correlazione tra la grandezza da stimare **E** ed una funzione **f**, secondo la forma seguente:

$$\mathbf{E = f\,(Vi\,)}$$

dove Vi sono variabili indipendenti (ad es. LOC e FP)
La struttura di questi modelli è di solito:

$$\mathbf{E = a + b \cdot (Vi\,)^{C}}$$

dove **a**, **b** e **c** sono costanti ricavate empiricamente.
I modelli sono ottenuti attraverso una analisi di regressione su dati dei progetti passati.

Ad esempio, sono modelli basati sulle LOC:

$E = 5.2 \cdot (KLOC)^{0.91}$	*Walston - Felix*
$E = 5.5 + 0.73 \text{ x } (KLOC)^{1.16}$	*Bailey - Basile*
$E = 3.2 \cdot (KLOC)^{1.05}$	*Boehm semplice*
$E = 5.288 \cdot (KLOC)^{1.047}$	*Doty (per Kloc >9)*

Sono modelli basati sui FP:

$E = 13.39 + 0{,}0545\ FP$	*Albrecht - Gaffey*
$E = 60.62 \cdot 7.728 \cdot 10^{-8} \cdot FP^{3}$	*Kemerer*
$E = 585.7 + 15.12 \cdot FP$	*Mattson-Barnett*

I modelli statistici/empirici funzionano come delle black-box che calcolano delle grandezze a partire da dati in ingresso:

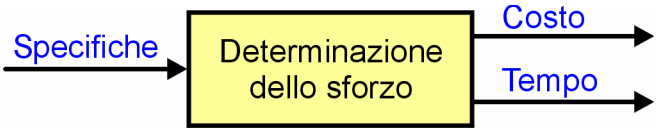


Figura 3: Modelli Statistici/Empirici come Black-box

I modelli statistici/empirici hanno in genere dei fattori di aggiustamento e regolazione per adattare i valori risultanti al contesto (alle specifiche del progetto).

L'esame dei diversi modelli proposti mostra che portano a risultati sempre diversi tra loro, è necessario quindi calibrarli.

8.2.1. Walston e Felix

Un eccellente esempio di modello statico a variabile singola fu proposto da Claude Walston e Chuck Felix al *Federal System Division* dell'IBM. Essi misero in relazione la dimensione con lo sforzo secondo la seguente equazione:

$$\mathbf{E = a \cdot L^{b}}$$

dove **E** è lo sforzo in mesi-uomo (MM), **L** rappresenta il numero di linee di codice in migliaia (KLOC), **a** e **b** sono due costanti il cui valore è stato ottenuto attraverso una regressione lineare sui dati ricavati da 60 progetti di vario genere. I progetti presi in esame variavano da 4K a 467K LOC, da 12 a 11758 mesi-uomo, svolgevano diversi compiti ed erano implementati con linguaggi diversi.

I valori di a e b così ottenuti sono:

a = 5.2
b = 0.91

Walston e Felix trovarono inoltre altre relazioni derivate:

$\mathbf{E = 5.2 \cdot L^{0.91}}$	Sforzo in mesi-uomo.
$\mathbf{DOC = 49 \cdot L^{1.01}}$	Pagine di Documentazione.
$\mathbf{D = 4.1 \cdot L^{0.36} = 2.47 \cdot E^{0.35}}$	Durata del progetto in mesi.
$\mathbf{S = 0.54 \cdot E^{0.6} = E/D}$	Dimensione media dello staff.

Per meglio comprendere l'influenza di una certa quantità di variabili sul processo di stima, Walston e Felix esaminarono 68 variabili in relazione con il loro effetto sulla produttività. Di queste 68 variabili esaminate solo 29 mostrarono un'elevata correlazione con la produttività (Tabella 5).

Al fine di stimare lo sforzo, queste 29 variabili furono combinate in un coefficiente di produttività (I – Productivity Index):

$$I = \sum_{i=1}^{29} W_i X_i$$

dove:

W_i = peso della domanda ($1/2 \cdot \log_{10}$[*rapporto del cambiamento di produttività per la domanda i*]).

X_i = risposta alla domanda (+1, 0, -1 a seconda se la risposta indica un incremento, nominale, decremento).

L'indice di produttività è stato usato per correggere la stima iniziale dell'equazione di partenza attraverso l'espressione delle deviazioni dalla norma.

Variabili significative:

Domanda o Variabile	Produttività Media (DLS/MM)			Variazione (DLS/MM)
Complessità dell'interfaccia cliente	< normale 500	normale 295	> normale 124	376
Partecipazione dell'utente nella definizione dei requisiti	nessuna 491	poca 267	molta 205	286
Variazioni di progetto generate dal cliente	poche 297		molte 196	101
Esperienza del cliente nell'area applicativa del progetto	nessuna 313	poca 340	molta 206	112
Esperienza e qualificazione totale del personale	bassa 132	media 257	alta 410	278
Percentuale dei programmatori che partecipano alle specifiche funzionali	< 25% 153	25% - 50% 242	> 50% 391	238
Esperienza precedente con computer operazionali	minima 146	media 270	intensiva 312	166
Esperienza precedente con i linguaggi di programmazione impiegati	minima 122	media 225	intensiva 385	263
Esperienza precedente con applicazioni simili per dimensione e complessità	minima 146	media 221	intensiva 410	264
Rapporto tra dimensione ideale dello staff e durata del progetto (mesi-uomo)	< 0.5 305	0.5 – 0.9 310	> 0.9 173	132
Hardware conforme allo sviluppo parallelo	si 297		no 177	120
Accesso allo sviluppo del computer, aperto sotto richiesta speciale	0% 226	1% - 25% 274	> 25% 357	131
Accesso allo sviluppo del computer, chiuso	0% - 10% 303	10% - 85% 251	> 85% 170	133
Ambiente classificato sicuro per i computer ed il 25% dei programmi e dati	no 289		si 156	133
Programmazione strutturata	0% - 33% 169	34% - 66% -	66% 301	132
Ispezione del progetto e del codice	0% - 33% 220	34% - 66% 300	> 66% 339	119
Sviluppo Top-Down	0% - 33% 196	34% - 66% 237	> 66% 321	125
Utilizzo di un capo per il team di programmatori	0% - 33% 219	34% - 66% -	> 66% 408	189
Complessità generale dello sviluppo del codice	< media 314		> media 185	129
Complessità del processo delle applicazioni	< media 349	media 345	> media 168	181
Complessità del flusso di programma	< media 289	media 299	> media 209	80
Vincoli totali del progetto	minimi 293	medi 286	elevati 166	127
Vincoli del progetto sulla memoria principale	minimi 391	medi 277	elevati 193	198
Vincoli del progetto sul tempo	minimi 303	medi 317	elevati 171	132
Codice per operazioni real-time o interattive o esecuzione in tempi molto limitati	< 10% 279	10% - 40% 337	> 40% 203	76
Percentuale di codice per la distribuzione	0% - 90% 159	91% - 99% 327	100% 265	106
Codice classificato come applicazioni non matematiche e programmi di formattazione I/O	0% - 33% 188	34% - 66% 311	67% - 100% 267	79
Numero di classi di entità nel database per 1000 linee di codice	0-15 334	16-80 243	> 80 193	141
Numero di pagine di documentazione prodotte per 1000 linee di codice prodotto	0-32 320	33-88 252	> 88 195	125

Tabella 7: variabili significativamente correlate con la produttività all'IBM/FSD

8.3. Modelli Analitici

8.3.1. Putnam

Larry H. Putnam ha sviluppato un modello di stima basato sull'assunzione che l'utilizzazione delle risorse umane per progetti tipici sia descritta dalla curva di Rayleigh ($y = a \cdot t \cdot e^{-a \cdot t^2}$) (fig. 4):

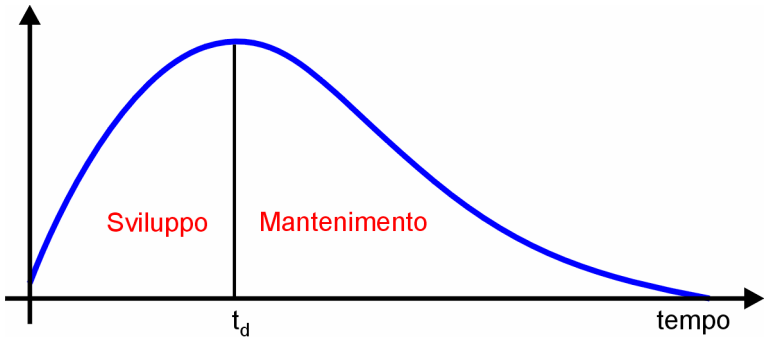


Figura 4: Curva di Rayleigh

Il modello derivato da Putnam prende il nome di "equazione del software":

$$B = \frac{L^3}{T^4 C^3}$$

$E = 0.3945 \cdot B$

dove:

- **B** = sforzo totale (intero ciclo di vita) espresso in anni-uomo.
- **E** = sforzo di sviluppo espresso in anni-uomo.
- **T** = tempo di sviluppo (in anni).
- **L** = dimensione espressa in LOC.
- **C** = costante tecnologica (dipende dalla tecnologia utilizzata; C=2000 – scarsa; C=8000 – buona; C=11000 - eccellente).

La curva di Putnam-Norden-Rayleigh (PNR) (fig. 5) mostra che il tempo di realizzazione del progetto può essere compresso o espanso all'interno di un range che va dal 75% al 200%. Le compressioni maggiori del 75% non sono attuabili senza modificare alcune delle caratteristiche fondamentali del progetto (volume, ambiente, …). Le espansioni maggiori del 200% hanno come effetto collaterale un aumento dei costi e non sono quindi praticabili.

Se esistono delle limitazioni sui tempi di realizzazione del progetto che sono diverse dal tempo ottimale (t_0) sarà necessario aggiustare sia i tempi che i costi.

t_a = tempo di consegna desiderato
t_d = tempo di consegna nominale

t_0 = tempo di consegna ottimale (in termini di costo) = $2t_d$
M_a = valore corretto di mesi-uomo

$$M_a = M \frac{t_d^4}{t_a^4}$$

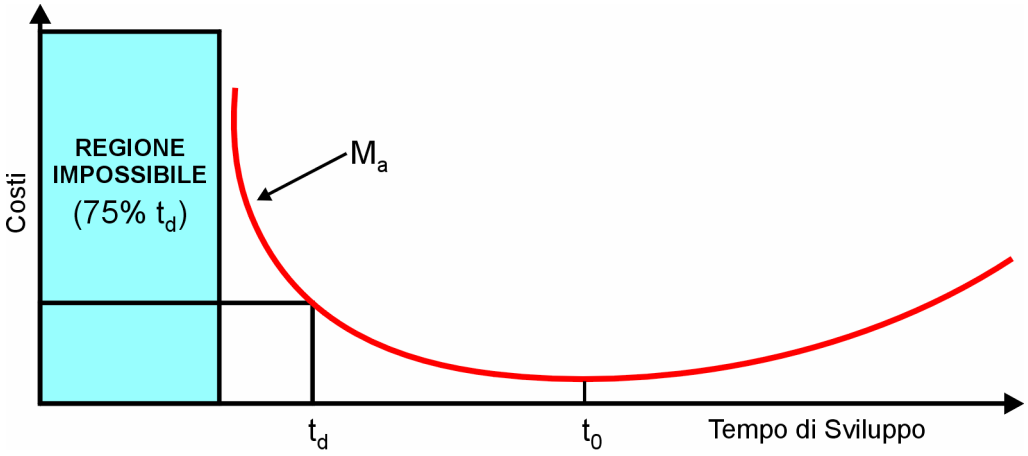


Figura 5: Curva di Putnam-Norden-Rayleigh (PNR)

Riducendo i tempi rispetto al valore calcolato **t_d** attraverso l'aggiunta di personale al progetto, si accelerano i tempi di consegna al costo di una diminuzione dell'efficienza fino al limite della *regione impossibile*. Oltre questo limite (75%), aggiungendo ancora personale si ottiene un aumento anziché una diminuzione dei tempi di consegna. Inversamente, aumentando i tempi oltre **t_d** si ha una riduzione del personale ed un'aumento dell'efficienza fino al limite del 200% oltre il quale i costi ed i tempi di consegna aumentano in maniera non accettabile. Questi costi aumentano a causa della rotazione del personale, il cambiamento dei requisiti, modifiche nello sviluppo e delle condizioni ambientali.

8.4. Modelli Compositi

8.4.1. COCOMO - Note sulla nomenclatura:

Al modello originale pubblicato nel 1981 dal Dr. Barry W. Boehm fu dato il nome di COCOMO. Questo nome è l'acronimo derivato dalle prime due lettere di ogni parola della frase ***Co****nstructive* ***Co****st* ***Mo****del* (Modello Costruttivo di Costo). La parola "costruttivo" si riferisce al fatto che il modello aiuta l'estimatore a capire meglio la complessità del lavoro software da svolgere rendendolo cosciente del perché il modello fornisce una particolare stima.

Al nuovo modello, seguente al modello COCOMO del 1981, fu inizialmente dato il nome di COCOMO 2.0 ma, per evitare confusioni con le implementazioni software del modello, il nome fu cambiato permanentemente in COCOMO II. Di conseguenza fu cambiato anche il nome del modello originale in COCOMO 81. I riferimenti bibliografici a COCOMO precedenti al 1995 si riferiscono a quello che oggi viene chiamato COCOMO 81. Molti dei riferimenti bibliografici pubblicati dopo il 1995 si riferiscono invece al COCOMO II.

Se esaminando della documentazione non si è sicuri a quale modello di COCOMO si fa riferimento, occorre controllare alcune parole chiave presenti nel testo:

Se vengono usati i termini *Basic*, *Intermediate*, or *Detailed* per i nomi di modello e *Organic*, *Semidetached*, or *Embedded* per modi di sviluppo, allora il modello a cui ci si riferisce è COCOMO 81.

Se invece vengono usati i termini *Application Composition*, *Early Design*, *Post-architecture*, o si citano i *scale factors (fattori di scala)* come *Precedentedness*, *Development Flexibility*, *Architecture/Risk Resolution*, *Team Cohesion*, *Process Maturity*, allora il modello a cui ci si riferisce è COCOMO II.

8.4.2. COCOMO 81

Il modello originale di COCOMO (COCOMO 81) fu sviluppato del Dr. Barry W. Boehm nel 1981 per fornire una stima del numero di uomini-mese (**E**) ed il tempo (**T**) necessari per lo sviluppo di un prodotto software a partire dalle specifiche. Questo modello assume un ciclo di vita a cascata (waterfall) ed è basato su uno studio di 63 progetti presso il TWR, una compagnia dove furono realizzati grandi progetti software. I programmi esaminati variavano da una dimensione di 2000 a 100000 LOC ed i linguaggi utilizzati variavano dall'assembly al PL/I.

Assunzioni:

- **E** è lo sforzo di sviluppo di un sistema software espresso i uomini-mese (MM) dove un mese equivale a 19 giorni lavorativi pari a 152 ore. Comprende gli sforzi di gestione e documentazione ma non quelli di istruzione, installazione, ecc.
- **T** è il tempo di sviluppo del software espresso in mesi. **T** inizia quando il progetto entra nella fase di progettazione del prodotto (successivamente alla fase di specifica dei requisiti) e finisce al completamento della fase di testing.
- I requisiti devono essere abbastanza stabili, ovvero non dovrebbero cambiare dopo la fase di specifica dei requisiti.

Vediamo ora COCOMO 81 nel dettaglio:

Definisce 3 livelli di stima (in base alla quantità di informazioni disponibili):

- **Basic:** utilizzato per stime approssimate veloci e premature, la sua accuratezza è limitata dal non utilizzo di dati dettagliati.
- **Intermediate:** fornisce una stima più accurata grazie alla considerazione di fattori dell'ambiente del progetto software in termini del loro effetto complessivo sui parametri del progetto.
- **Detailed:** fornisce la miglior stima in quanto tiene conto dei fattori dell'ambiente del progetto software in ogni singola fase del progetto.

Identifica 3 tipi di progetto cui si applica (in base alla complessità del progetto):

- **Organic:** Applicazioni semplici e di limitate dimensioni, requisiti poco stringenti, ambiente di sviluppo noto.
- **Semidetached:** Complessità e dimensioni medie. Applicazioni comprese tra il modello *Organic* ed *Embedded*.
- **Embedded:** Applicazioni complesse, con attento controllo del processo e rigidi vincoli sulla qualità (i.e. sistemi real time, applicazioni militari o per il volo).

Basic

Fornisce una precisione di ±20% nel 60% dei casi.

Lo sforzo **E** richiesto per lo sviluppo di un sistema software è funzione della dimensione del codice **S** espressa in KLOC (migliaia di linee di codice):

$$E = a \cdot S^b$$

La durata **T** del progetto è funzione dello sforzo **E**:

$$T = c \cdot E^d$$

I valori delle costanti **a**, **b**, **c**, **d,** variano a seconda dei tipi di progetto secondo la seguente tabella:

Tipo di progetto	a	b	c	d
Organic	2.40	1.05	2.50	0.38
Semidetached	3.00	1.12	2.50	0.35
Embedded	3.60	1.20	2.50	0.32

Tabella 8: Valori delle costanti per il modo Basic

Intermediate

Fornisce una precisione di ±20% nel 68% dei casi.

Questo modello è basato sul modello Basic al quale introduce 15 fattori correttivi detti Fattori di Complessità (*Cost Drivers*) o Moltiplicatori di Sforzo (EM – Effort Multipliers). Ad ogni fattore di costo occorre assegnare un valore secondo la seguente tabella:

Fattore di Complessità (Cost Driver)	Molto Bassa (Very Low)	Bassa (Low)	Nominale (Nominal)	Alta (High)	Molto Alta (Very High)	Altissima (Extra High)
Attributi di Prodotto (Product Attributes)						
RELY Required Software Reliability Affidabilità richiesta intesa come difettosità residua	0.75	0.88	1.00	1.15	1.40	-
DATA Database Size Dimensione relativa al programma dei dati gestiti	-	0.94	1.00	1.08	1.16	-
CPLX Product Complexity Complessità globale del prodotto	0.70	0.85	1.00	1.15	1.30	1.65
Attributi del Computer (Computer Attributes)						
TIME Execution Time Constraint Requisiti di efficienza	-	-	1.00	1.11	1.30	1.66
STOR Main Storage Constraint Requisiti di memoria centrale	-	-	1.00	1.06	1.21	1.56
VIRT Virtual Machine Volatility Frequenza con cui le caratteristiche del sistema target vengono modificate durante lo sviluppo	-	0.87	1.00	1.15	1.30	-
TURN Computer Yurnaround Time Tempo di risposta accettabile	-	0.87	1.00	1.07	1.15	-
Attributi del Personale (Personnel Attributes)						
ACAP Analyst Capability Efficienza del personale di analisi	1.46	1.19	1.00	0.86	0.71	-
AEXP Applications Experience Esperienza nello sviluppo di sw simile	1.29	1.13	1.00	0.91	0.82	-
PCAP Programmer Capability Efficienza del personale di programmazione	1.42	1.17	1.00	0.86	0.70	-
VEXP Virtual Machine Experience Disponibilità di esperienza nell'uso della macchina target	1.21	1.10	1.00	0.90	-	-
LEXP Language Experience Esperienza nell'uso del linguaggio di programmazione	1.14	1.07	1.00	0.95	-	-
Attributi del Progetto (Project Attributes)						
MODP Modern Programming Practices Uso di tecniche di programmazione efficienti	1.24	1.10	1.00	0.91	0.82	-
TOOL Use of Software Tools Uso di strumenti automatizzati	1.24	1.10	1.00	0.91	0.83	-
SCED Required Development Schedule Vincoli sul tempo di conclusione del progetto	1.23	1.08	1.00	1.04	1.10	-

Tabella 9: Moltiplicatori di Sforzo (EM - Cost Drivers)

Il prodotto dei valori assegnato a ciascuno dei 15 fattori di complessità fornisce il valore correttivo (EAF):

$$EAF = \prod_{i=1}^{15} EM_i$$

dove $EM_{1,...,15}$ sono i valori assegnati ai fattori di complessità (moltiplicatori di sforzo) secondo la tabella 9.

Lo sforzo **E** richiesto per lo sviluppo di un sistema software è funzione dello sforzo nominale $\mathbf{E_{nom}}$:

$$E = E_{nom} \cdot EAF$$

Lo sforzo nominale $\mathbf{E_{nom}}$ (come lo sforzo **E** nel modello Basic) è funzione della dimensione del codice **S** espressa in KLOC (migliaia di linee di codice):

$$E_{nom} = a \cdot S^b$$

La durata **T** del progetto è funzione dello sforzo **E**:

$$T = c \cdot E^d$$

I valori delle costanti **a**, **b**, **c**, **d,** variano a seconda dei tipi di progetto secondo la seguente tabella:

Tipo di progetto	a	b	c	d
Organic	3.20	1.05	2.50	0.38
Semidetached	3.80	1.12	2.50	0.35
Embedded	2.80	1.20	2.50	0.32

Tabella 10: Valori delle costanti per il modo Intermediate

Detailed

Fornisce una precisione di ±20% nel 70% dei casi.
A differenza del modello *Intermediate*, il modello *Detailed* usa differenti *Fattori di Complessità* (EAF) per ogni fase del progetto.

COCOMO Detailed definisce sei fasi del ciclo di vita:

- **RQ**: requisiti (requirements)
- **PD**: progettazione prodotto (product design)
- **DD**: progettazione dettagliata (detailed design)
- **CT**: codifica e test di unità (coding and unit testing)
- **IT**: integrazione e test (integration and testing)
- **MN**: mantenimento (manteinance)

Le fasi PD, DD, CT e IT sono chiamate *fasi di sviluppo*. Le stime per la fase dei requisiti (RQ) e del mantenimento (MN) vengono realizzate in modo differente dalla fase di sviluppo.

Per rendere più agevole il procedimento di stima, il programma viene organizzato secondo una gerarchia a 3 livelli:

- Modulo (Module)
- Sottosistema (Subsystem)
- Sistema (System)

I *Fattori di Complessità* (EAF) variano da sottosistema a sottosistema ma tendono ad essere gli stessi per tutti i moduli all'interno dello stesso sottosistema.

Il modello *Detailed* illustra l'importanza di riconoscere diversi livelli di previsione ad ogni fase del ciclo di sviluppo del software. Boehm ebbe qui una buona idea ma COCOMO 81 da solo non era un modello sufficientemente robusto per predire accuratamente i costi di tutte le fasi di sviluppo. Provare ad applicare i pesi durante la fase di analisi dei requisiti quando i dati necessari per la stima non sono sufficientemente accurati (almeno fino alla fase di progetto), mostra tutte le limitazioni di questo modello.

8.4.3. COCOMO II

Verso la metà degli anni novanta le tecniche di sviluppo del software cambiarono drasticamente. Questi cambiamenti includevano modelli di sviluppo rapidi e non sequenziali, l'enfatizzazione del riuso del software esistente, l'utilizzo di componenti software standardizzate e pronte all'uso, reingegnerizzazione, l'*applications composition*, l'approccio object-oriented.

Questi cambiamenti resero problematica l'applicazione di COCOMO 81, così la soluzione più logica fu quella di reinventare il modello per gli anni novanta. Dopo diversi anni di sforzi combinati tra *USC (University of Southern California) – CSE (Center for Software Engineering), IRUS* presso *UC Irvine* ed il *COCOMO II Project Affiliate Organizations*, il risultato fu COCOMO II, un modello rivisitato per la stima dei costi del software che rifletteva i cambiamenti avvenuti nello sviluppo professionale di software fin dagli anni settanta.

COCOMO II, diversamente da COCOMO 81 che usa solo il ciclo di vita a cascata (waterfall), deve adattarsi ai diversi cicli di vita del software ed alle particolarità dei loro processi (disponibilità di software riusabile, grado di comprensione dell'architettura e dei requisiti, vincoli di tempo o di dimensione, affidabilità richiesta,…).

La **granularità** (*granularity*) della stima deve essere consistente rispetto alla granularità delle informazioni disponibili, pertanto COCOMO II fornisce stime a granularità larga nelle fasi iniziali del progetto e stime a granularità sempre più fine con l'avanzare delle fasi (fig. 6).

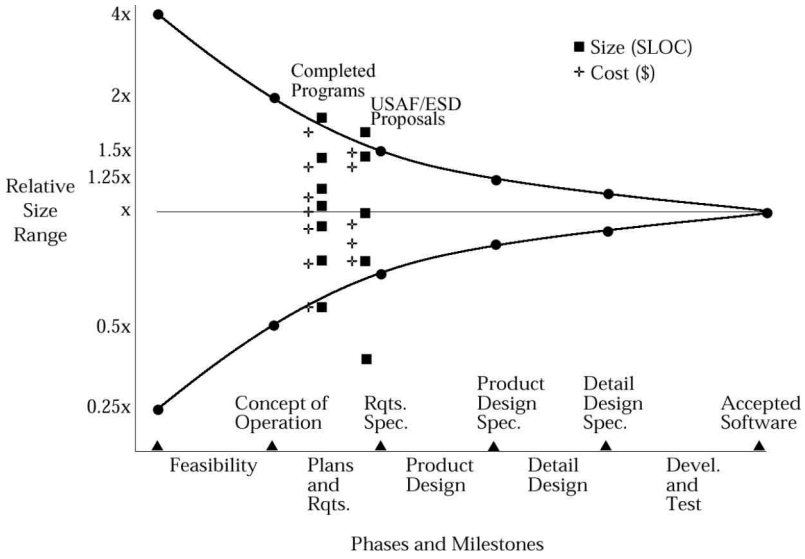


Figura 6: Precisione della stima dei costi e delle dimensioni rispetto alla fase del ciclo di vita

COCOMO II divide il mercato software in 5 segmenti principali:

- **End-User Programming (95.24%):** programmi piccoli e flessibili, sviluppati dagli stessi utenti attraverso degli *Application Generators.*
- **Infrastructure (1.36%):** prodotti nell'area dei sistemi operativi, sistemi di gestione dei database, sistemi di networking. I produttori di questo tipo di software, contrariamente ai programmatori end-user, hanno un'ottima conoscenza della *computer science* ed una conoscenza scarsa delle applicazioni.
- **Application Generators and Composition Aids (1.09%):** prodotti *prepackaged* per *user programming*. Questi prodotti hanno molti componenti riusabili ma richiedono comunque ottime capacità di programmazione.
- **Application Composition (1.27%):** applicazioni troppo diversificate per essere gestite da soluzioni *prepackaged* ma che sono sufficientemente semplici da poter essere composte da componenti. Componenti tipici sono *graphic user interface (GUI) builders*, *database* o *object managers*, *middleware* per processi distribuiti o processi transazionali, gestori hypermediali, *smart data finders*, componenti *domain-specific* come finanziari, medici, o controllo di processi industriali.
- **System Integration (1.27%):** sistemi a larga scala, sistemi ad alta integrazione o sistemi senza precedenti. Porzioni di questi software possono essere sviluppate tramite *Application Composition* ma generalmente richiedono una programmazione dedicata.

Il settore *End-User Programming* non necessita di COCOMO II come modello di stima in quanto questo tipo di applicazioni vengono generate in ore o giorni, pertanto una stima basata sull'attività è più che sufficiente.

Il modello COCOMO II per *Application Composition* è basato sugli **Object Points**.

La stima COCOMO II per i settori *Application Generators, System Integration* o *Infrastructure* è basata su una combinazione del modello per l'*Application Composition* (per gli sforzi di prototipazione rapida) e due modelli di stima incrementali per due successive porzioni del ciclo di vita. L'appropriata sequenza di questi modelli dipenderà da fattori di mercato del progetto e dal suo grado di comprensibilità.

I tre modelli relativi all'*Application Generators*, *System Integration* e *Infrastructure* sono:

- **Application Composition:** modello che include gli sforzi di prototipazione per evitare elevati rischi potenziali come le interfacce utente, interazioni software/sistema, prestazioni, maturità tecnologica.
- **Early Design:** modello per la fase iniziale della progettazione (esplorazione alternativa di architetture software, di sistema e concetti operativi). A questo punto non si hanno sufficienti informazioni per ottenere una stima a granularità fine. Questo modello fa uso di *Punti Funzione* (FP - *Function Points*) e un ristretto numero addizionale di *coefficienti di sforzo* (*cost drivers*).
- **Post-Architecture:** modello per la fase di sviluppo ed il mantenimento del software. Fornisce stime precise se l'architettura del ciclo di vita è stata ben definita rispetto gli obbiettivi del sistema, le operazioni concettuali ed i rischi. Questo modello ha la stessa granularità dei precedenti modelli COCOMO 81 e Ada COCOMO. Fa uso di linee codice sorgente (SLOC) e/o di Punti Funzione (FP) come metrica dimensionale, coefficienti per il riuso ed i guasti software (*software breakage*), 17 moltiplicatori di sforzo e 5 fattori che determinano l'esponente di scala del progetto (project's scaling exponent). Questi fattori sostituiscono i modi di sviluppo (Organic, Semidetached, Embedded) nel modello COCOMO 81 e perfezionano i quattro esponenti di scala di Ada COCOMO.

COCOMO II prevede l'uso di tre unità di misura per la dimensione:

- **Object Points**
- **Unadjusted Function Points** (secondo la definizione IFPUG – *International Function Points User Group*)
- **Source Lines Of Code** (SLOC – secondo la definizione del SEI – *Software Engineering Institute*)

COCOMO II utilizza un'equazione del riuso (reuse equation - linee di codice equivalenti al nuovo software da sviluppare) che corregge la non linearità del vecchio modello utilizzato in COCOMO 81.

$$ESLOC = ASLOC \cdot \frac{AA + SU + 0.4 \cdot DM + 0.3 \cdot CM + 0.3 \cdot IM}{100}$$

dove:

- **ESLOC – Equivalent SLOC:** linee di codice equivalenti alle nuove istruzioni.
- **ASLOC – Adapted SLOC:** linee di codice di software da adattare
- **AA – Assessment and Assimilation:** valutazione ed integrazione. Il valore è ricavato dalla tabella 11.
- **SU – Software Understanding:** Comprensibilità del software. Il valore è ricavato dalla tabella 12.
- **DM - Design Modification:** percentuale di modifiche al progetto
- **CM - Code Modification:** percentuale di codice modificato
- **IM – Integration Effort:** percentuale di sforzo di integrazione originale necessario per integrare il software di riuso.

Incremento AA	Livello di Sforzo AA
0	Nessuno
2	Ricerca moduli base e documentazione
4	Test e valutazione di alcuni moduli documentazione
6	Test e valutazione di numerosi moduli, documentazione
8	Test e valutazione intensiva dei moduli, documentazione

Tabella 11: scala per il calcolo dell'AA

	Molto Bassa	Bassa	Nominale	Alta	Molto Alta
Struttura (Structure)	coesione molto bassa, alto accoppiamento.	coesione moderatamente bassa, alto accoppiamento.	considerevolmente ben strutturato; alcune aree deboli.	alta coesione, basso accoppiamento.	Forte modularità, occultamento dell'informazione nei dati e nelle strutture di controllo.
Chiarezza dell'Applicazione (Application Clarity)	Nessuna correlazione tra il punto di vista dell'applicazione e del programma	Alcune correlazioni tra il programma e l'applicazione.	Moderata correlazione tra programma e applicazione.	Buona correlazione tra programma e applicazione.	Chiara correlazione tra i punti di vista del programma e l'applicazione.
Autodescrittività (Self-Descriptiveness)	Codice oscuro; documentazione mancante, incomprensibile o obsoleta.	Pochi commenti sul codice e intestazioni. Poca documentazione utile	Moderato livello di commenti sul codice, intestazioni e documentazione	Buoni commenti e intestazioni. Documentazione utile. Alcune aree deboli.	Codice autodescrittivo; documentazione aggiornata, ben organizzata e razionale.
incremento SU a AAF (SU Increment to AAF)	50	40	30	20	10

Tabella 12: Scala per il calcolo dell'SU

COCOMO II considera la volatilità dei requisiti attraverso il Moltiplicatore di Sforzo della Volatilità dei Requisiti (**BRAK** – Requirements Volatility effort multiplier). Questo parametro viene utilizzato per correggere l'effettiva dimensione del prodotto ma non viene utilizzato nel modello *Application Composition* che incorpora già un meccanismo di correzione.

Il parametro ACT – Annual Change Traffic (percentuale annuale di codice aggiunto e modificato) presente in COCOMO 81 è stata sostituita con il modello del riuso che gestisce ottimamente anche il mantenimento.

Lo sforzo nominale (**E_{nom}**) richiesto per lo sviluppo di un sistema software (espresso in MM: uomini-mese) è dato dalla seguente formula:

$$E_{nom} = A \cdot (Size)^B$$

dove:

- **A** = costante usualmente pari a 2.94
- **Size** = dimensione del codice in KSLOC
- **B** = esponente del fattore di scala che tiene conto di alcuni elementi di complessità del progetto che provocano delle diseconomie di scala.

Per il modello *Application Composition* B vale sempre 1, mentre per i modelli *Early Design* e *Post Architecture* varia tra 1.01 e 1.26 secondo la formula:

$$B = 1.01 + 0.01 \sum_{i=1}^{5} W_i$$

dove W_i sono i valori assegnati a 5 fattori di scala secondo la seguente tabella:

Fattori di Scala	**Molto Bassa**	**Bassa**	**Nominale**	**Alta**	**Molto Alta**	**Altissima**
W_i	5	4	3	2	1	0
PREC Precedenti	Completamente senza precedenti	largamente senza precedenti	qualche precedente	generalmente noto	largamente noto	completamente noto
FLEX Flessibilità dello sviluppo	rigorosa	occasionalmente rilassata	poco rilassato	conformità generale	poca conformità	obiettivi generali
RESL Architettura / risoluzione rischi (%)	pochi (20%)	alcuni (40%)	spesso (60%)	in genere (75%)	prevalente (90%)	sempre (100%)
TEAM Coesione del team	interazioni molto difficili	alcune difficoltà di interazione	interazioni fondalmentame nte cooperative	largamente cooperative	altamente cooperative	interazioni perfette
PMAT Maturità del processo	Pesato rispondendo "SI" al questionario di Maturità del CMM (Capability Maturity Model).					

Tabella 13: Fattori di Scala

Così come nel modello COCOMO 81, anche COCOMO II corregge il Valore dello sforzo attraverso una produttoria dei Moltiplicatori di Sforzo (EM – Effort Multipliers) (tab.14):

$$E = E_{nom} \cdot \prod_i EM_i$$

Fattore di Complessità (Cost Driver)	Molto Bassa (Very Low)	Bassa (Low)	Nominale (Nominal)	Alta (High)	Molto Alta (Very High)	Altissima (Extra High)
Attributi di Prodotto (Product Attributes)						
RELY Required Software Reliability Affidabilità richiesta intesa come difettosità residua	Leggeri inconvenienti	basse perdite facilmente recuperabili	perdite moderate facilmente recuperabili	grandi perdite finanziarie	rischio per la vita umana	
DATA Database Size Dimensione relativa al programma dei dati gestiti		DB bytes/Pgm SLOC<10	10≤D/P<100	100≤D/P<1000	D/P≥1000	
CPLX Product Complexity Complessità globale del prodotto	Calcolato in base ad un'altra tabella					
RUSE Required Reusability Riuso richiesto		nessuno	di progetto	di programma	di linea produttiva	di più linee produttive
DOCU Documentation match to life-cycle needs Livello di documentazione richiesta durante il ciclo di vita	Molte parti del ciclo di vita scoperte	Alcune parti del ciclo di vita scoperte	tutte le parti del ciclo di vita coperte in giusta misura	Copertura eccessiva	Copertura estremamente eccessiva	
Attributi del Computer (Computer Attributes)						
TIME Execution Time Constraint Requisiti di efficienza			uso del tempo di esecuzione disponibile ≤ 50%	70%	85%	95%
STOR Main Storage Constraint Requisiti di memoria centrale			uso della memoria disponibile ≤ 50%	70%	85%	95%
PVOL Platform Volatility Cambiamento delle caratteristiche della piattaforma		cambiamenti maggiori: 12 mesi; minori: 1 mese	cambiamenti maggiori: 6 mesi minori: 2 settimane	cambiamenti maggiori: 2 mesi minori: 1 settimane	cambiamenti maggiori: 2 settimane minori: 1 giorno	
Attributi del Personale (Personnel Attributes)						
ACAP Analyst Capability Efficienza del personale di analisi	15° percentuale	35° percentuale	55° percentuale	75° percentuale	90° percentuale	
PCAP Programmer Capability Efficienza del personale di programmazione	15° percentuale	35° percentuale	55° percentuale	75° percentuale	90° percentuale	
PCON Personnell Continuity Cambiamento annuale del personale	48%/anno	24%/anno	12%/anno	6%/anno	3%/anno	
AEXP Applications Experience Esperienza nello sviluppo di sw simile	≤ 2 mesi	6 mesi	1 anno	3 anni	6 anni	
PEXP Platform Experience Esperienza nella piattaforma utilizzata	≤ 2 mesi	6 mesi	1 anno	3 anni	6 anni	
LTEX Language and Tool Experience Esperienza nel linguaggio e negli strumenti utilizzati	≤ 2 mesi	6 mesi	1 anno	3 anni	6 anni	
Attributi del Progetto (Project Attributes)						
TOOL Use of Software Tools Uso di strumenti automatizzati	edit, code, debug	simple, frontend, backend CASE, little integration	basic lifecicle tools, moderately integrated	strong, mature lifecicle tools, moderately integrated	strong, mature, proactive lifecicle tools, well integrated with processes, methods, reuse	
SITE Collocation	Internazionale	Più città e più compagnie	Più città o più compagnie	alcune città o aree metropolitane	Alcuni edifici o complessi	Pienamente collocato
SITE Communications	Alcune telefonate, posta	Telefonate individuali, fax	email a banda stretta	comunicazioni elettroniche a banda larga	comunicazioni a banda large e videoconferenza occasionale	Comunicazioni interattive multimediali
SCED Required Development Schedule Vincoli sul tempo di conclusione del progetto	75% del nominale	85%	100%	130%	160%	

Tabella 14: Moltiplicatori di Sforzo (EM - Cost Drivers)

Nella versione originale di COCOMO II il tempo di sviluppo (**T**), espresso in mesi, è calcolato come:

$$T = [3.0 \cdot E^{(0.33+0.2\cdot(B-1.01))}] \cdot \frac{SCEDPercentuale}{100}$$

Le versioni successive hanno dei metodi di stima del tempo di sviluppo che riflettono le differenti classi di modello di processo che possono essere usate in un progetto.

9. Metriche Object-Oriented

9.1. Introduzione

Verso la fine degli anni ottanta l'introduzione della programmazione orientata agli oggetti (*object-oriented programming*) ha creato la necessità di metriche adatte a questo nuovo tipo di approccio ed alle sue caratteristiche (classi, eredità, riuso, incapsulamento e passaggio dei messaggi). L'introduzione del concetto di "oggetto" ha reso obsolete alcune metriche tradizionali che consideravano *dati* e *funzioni* come entità separate.

Contrariamente alle *metriche tradizionali* o *convenzionali*, lo studio di *metriche object-oriented* è quindi relativamente nuovo e non ha ancora raggiunto la sua piena maturità.

Le metriche object-oriented, oltre a fornire misure sulla struttura o il comportamento di un sistema object-oriented, costituiscono uno strumento essenziale per la gestione del software (management).

Le metriche che prenderemo in esame sono quelle maggiormente citate in letteratura e presenti anche in alcuni *object-oriented tools*. Esse si applicano ai concetti dell'object-oriented (metodi, classi, accoppiamento, eredità) e focalizzano la loro attenzione soprattutto:

- sulla **struttura interna dell'oggetto** che riflette la complessità di ogni entità individuale (come le classi e i metodi);
- sulla **complessità esterna** che misura le interazioni fra entità (come l'accoppiamento e l'eredità).

Le metriche object-oriented misurano inoltre la complessità computazionale che influisce:

- sull'efficienza di un algoritmo
- sull'utilizzo di risorse della macchina
- sui fattori di complessità psicologica che a loro volta influiscono sull'abilità di un programmatore nel creare, comprendere, modificare e curare la manutenzione del software.

9.2. Classificazione delle Metriche Object-Oriented

Attualmente non esiste una classificazione universalmente accettata.

Una prima valida classificazione può essere quella adottata da Mark Lorenz e Jeff Kidd nel libro “Object-Oriented Software Metrics”:

- **Project Metrics (Metriche di Pianificazione)**: metriche relative ai problemi manageriali. Esse sono divise in tre sezioni: *Application Size*, *Staffing Size*, *Scheduling*.
- **Design Metrics (Metriche di Progetto)**: metriche usate per quantificare la complessità, la dimensione e la robustezza di un progetto object-oriented.

Ad ogni metrica possono essere associati i seguenti attributi:

- **Name (nome)**: unico nome descrittivo per la metrica.
- **Meaning (significato)**: descrizione dell’informazione che la metrica fornisce all’utente.
- **Project Results (risultati di progetto)**: rappresentazioni grafiche dei dati statistici raccolti.
- **Affecting Factors (fattori influenzanti)**: riguarda l’interdipendenza fra le metriche e altri fattori nel progetto (come ad esempio l’interfaccia utente).
- **Related Metrics (metriche correlate)**: relazioni esistenti tra le metriche.
- **Thresholds (livelli di riferimento)**: soglie utilizzate per discriminare tra valori accettabili e inaccettabili.
- **Suggested Actions (azioni suggerite)**: suggeriscono agli utenti quali azioni eseguire nel caso in cui il valore della metrica si trovi fuori dal range consigliato.

9.2.1. Project Metrics (Metriche di Pianificazione)

Application Size

Le metriche di tipo *Application Size* sono state create con lo scopo di fornire al management una comprensione della quantità di lavoro necessaria per una specifica dell’applicazione. Siccome questa quantità di lavoro varia con la portata e con la complessità dell’applicazione, sono state scelte misure focalizzate soprattutto sugli obbiettivi principali dell’applicazione stessa.

Alcuni esempi di queste metriche sono *Number of Key Classes*, *Number of Subsystems*, e *Number of Support Classes*.

Chiaramente più grande è l’applicazione, maggiori saranno i valori di queste metriche.

Staffing Size

Le metriche di tipo *Staffing Size* sono legate alla dimensione e alla complessità dell'applicazione. A questo scopo sono state proposte due misure: *Person-Days Per Class* e *Classes Per Developer*.

Se applichiamo l'idea del *Goal-Question-Measure* (Grady and Caswell, 1987) a queste metriche, abbiamo la seguente progressione:

- *Goal*: Predire accuratamente il personale richiesto per completare un'applicazione "X"
- *Questions*: Quanto tempo impiega un programmatore medio a completare un'unità logica di codice (in questo caso una classe)? Quante unità logiche di codice (classi) possiamo assegnare a un programmatore?
- *Measure*: Le metriche proposte dagli autori rispondono abbastanza bene alle due domande.

Scheduling

Si riferiscono a software di tipo commerciale. Forniscono un utile strumento di pianificazione temporale dei processi nell'ambito della gestione dei progetti.

9.2.2. Design Metrics (Metriche di Progetto)

Le metriche di progetto object-oriented consentono di valutare la complessità di un sistema software nelle fasi iniziali del progetto, affinché le classi e gli oggetti troppo complessi possano essere riprogettati. Di conseguenza si ha un aumento della qualità dovuto alla riduzione della complessità.

Di seguito verranno illustrati due importanti gruppi di *metriche di progetto*:

CK - Chidamber e Kemerer

Queste metriche, comunemente indicate come *metriche CK*, furono introdotte da Chidamber e Kemerer nel 1991 e costituiscono uno dei contributi fondamentali al campo delle metriche object-oriented.

Le metriche CK sono 6 e si presentano con molteplici definizioni, in quanto i ricercatori e i professionisti non hanno raggiunto una definizione e un algoritmo di calcolo comuni. In alcuni casi, infatti, l'algoritmo di calcolo è influenzato dallo stesso software usato per calcolare le metriche.

Per alcune delle metriche descritte verrà dato un esempio in riferimento all'applicazione riportata nella figura seguente:

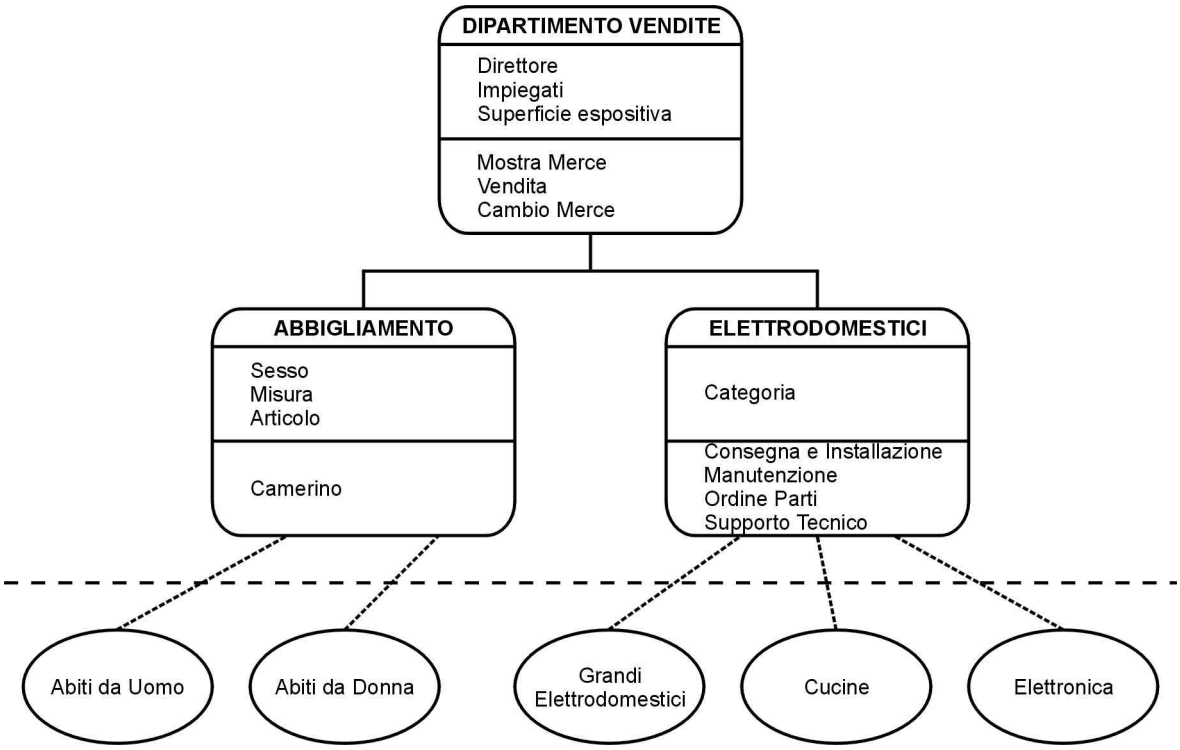


Figura 7: Esempio di un'applicazione object-oriented

WMC - Weighted Method per Class (Metodi Pesati per Classi)

La metrica *Weighted Method per Class* può essere definita in due modi: come il numero di metodi implementati nella classe, oppure come la somma delle complessità di tutti i metodi della classi (le complessità dei metodi sono calcolate con la *Cyclomatic Complexity*).

La seconda misura è difficile da implementare in quanto, a causa dell'eredità, non tutti i metodi sono accessibili all'interno della gerarchia di classe.

Il numero e la complessità dei metodi richiesti da questa metrica ci fanno capire quanto sforzo e quanto tempo sia necessario per curare la manutenzione di una classe. Più grande è il numero di metodi in una classe, più grande sarà l'impatto potenziale sui "figli", in quanto questi ereditano tutti i metodi definiti nella classe "padre". Le classi con un grande numero di metodi sono più specifiche per le applicazioni, limitando però la possibilità di riuso.

Questa metrica fornisce informazioni circa la comprensibilità, il riuso, e la manutenzione.

In riferimento alla figura 7 la WMC è data dal numero di metodi in ogni classe:

Per *Abbigliamento*, WMC = 1; per *Elettrodomestici*, WMC = 4

RFC - Response for a Class (Risposta per Classe)

La metrica *Response for a Class* è definita come la cardinalità (numero di elementi) dell'insieme di tutti i metodi che possono essere chiamati in risposta ad un messaggio ricevuto da un oggetto della classe o mandato da un metodo della classe stessa. Questo include tutti i metodi accessibili all'interno della gerarchia della classe.

Questa metrica usa il numero dei metodi per fornire informazioni circa la combinazione fra la complessità della classe e la possibilità di comunicazioni con le altre classi.

Più grande è numero di metodi che può essere chiamato da una classe attraverso messaggi, più grande sarà la complessità della classe.

Se in risposta ad un messaggio può essere chiamato un gran numero di metodi allora le operazioni di testing e debugging della classe richiederanno una maggiore capacità di comprensione da parte di colui che si occupa di tali operazioni.

Questa metrica misura la comprensibilità, la manutenzione e la collaudabilità.

In riferimento alla figura 7 la RFC per la classe *Dipartimento Vendite* è il numero di metodi che possono essere chiamati in risposta a messaggi inviati dalla stessa *Dipartimento Vendite*, dalla *Abbigliamento* e dalla *Elettrodomestici*. Dunque RFC(*Dipartimento Vendite*) = 3(se stessa) + 1(*Abbigliamento*) + 4(*Elettrodomestici*) = 8.

LCOM - Lack of Cohesion of Methods (Carenza di Coesione dei Metodi)

La metrica Lack of Cohesion of Methods misura il grado di diversità fra metodi di una classe in riferimento alle variabili o agli attributi (proprietà strutturali delle classi) utilizzati dagli stessi.

Una misura della separazione fra metodi aiuta ad identificare i difetti nella progettazione delle classi.

Esistono almeno due modi per misurare la coesione:

Per ogni campo dati in una classe, si calcola la percentuale di metodi che usano tale campo dati; poi si mediano le percentuali e si sottrae dal 100%.

Una percentuale bassa indica una grande coesione di dati e metodi fra le classi.

Si contano gli insiemi disgiunti prodotti dall'intersezione fra insiemi di attributi usati dal metodo. Infatti i metodi sono più simili se operano sugli stessi attributi.

Sia C una classe avente metodi M_1 , M_2 ,... , M_m e sia I_k l'insieme di variabili usate dal metodo M_k. Definiamo i seguenti insiemi:

$P = \{ (I_k, I_j) \mid I_k \cap I_j = 0 \}$, $Q = \{ (I_k, I_j) \mid I_k \cap I_j \neq 0 \}$

Se $| P | > | Q |$ allora $LCOM = | P | - | Q |$ altrimenti $LCOM = 0$

Nel seguito verrà utilizzata questa seconda definizione.

Un'alta coesione implica semplicità ed elevata possibilità di riuso e indica una buona suddivisione fra classi; un modulo con coesione molto alta potrebbe addirittura essere unico.

Una scarsa o bassa coesione aumenta la complessità e perciò aumenta la probabilità di errori durante il processo di sviluppo; probabilmente classi con bassa coesione possono essere suddivise in due o più sottoclassi con coesione maggiore.

Questa metrica misura l'efficienza e il riuso.

In figura 8 viene mostrata una rappresentazione alternativa della figura 7 che mette in evidenza l'utilizzo di tale metrica.

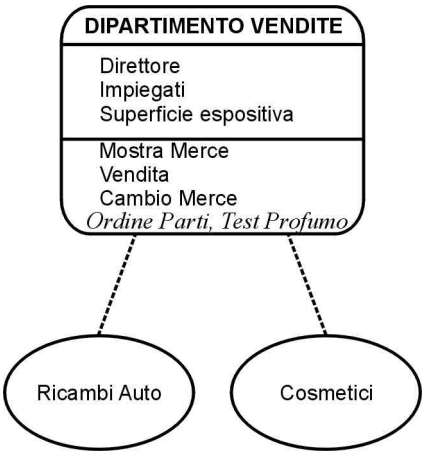


Figura 8: Esempio alternativo

CBO - Coupling Between Object Classes (Accoppiamento tra Classi)

La metrica *Coupling Between Object Classes* è definita come il numero di classi con la quale una data classe è accoppiata.

Un eccessivo accoppiamento è dannoso alla progettazione modulare e impedisce il riuso; infatti al crescere dell'indipendenza di una classe, cresce anche la facilità a riutilizzarla in altre applicazioni, mentre al crescere del numero di coppie, cresce la sensitività del progetto al modificarsi di altre parti e dunque la manutenzione e il riuso risultano più difficili.

Un forte accoppiamento può complicare un sistema, poiché una classe (modulo) è più difficile da comprendere, modificare o correggere se è messa in relazione con altre classi (moduli).

La complessità può essere ridotta progettando sistemi con un accoppiamento fra classi il più piccolo possibile. Questo migliora la modularità e favorisce l'incapsulamento.

Questa metrica misura l'efficienza e la riusabilità.

In figura 9 è mostrato un esempio (un po' esagerato) di elevato accoppiamento fra gli oggetti. Ci sono due dipartimenti *Giacche* e *Pantaloni* che hanno gli stessi attributi e gli stessi metodi. Questo implica una perdita di efficienza del progetto in quanto questi dipartimenti potrebbero essere combinati in un'unica classe.

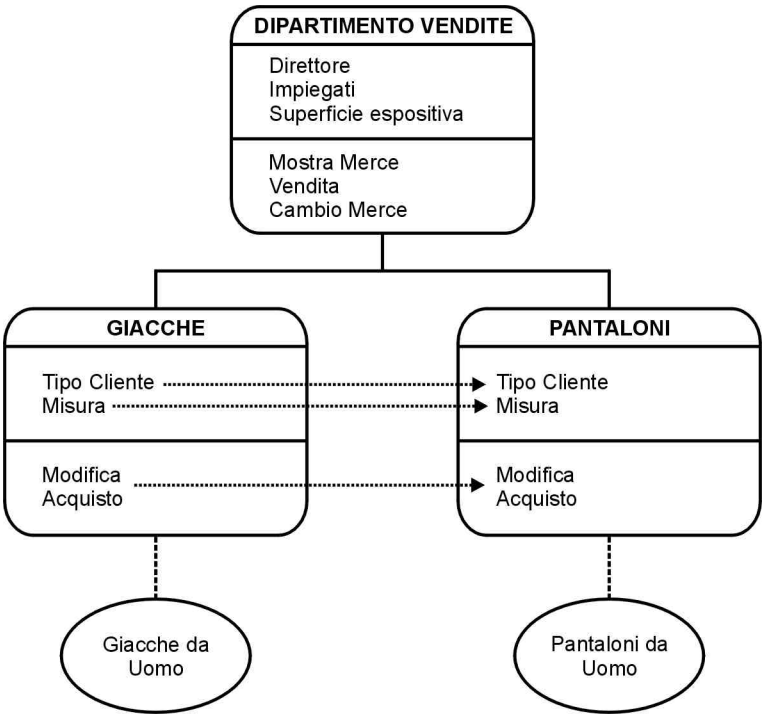


Figura 9: Esempio di eccessivo accoppiamento

DIT - Depth of Inheritance Tree (Profondità dell'Albero di Eredità)

La metrica *Depth of Inheritance Tree* rappresenta la profondità di una classe all'interno della gerarchia dell'eredità intesa come la distanza (numero di livelli) da un nodo classe alla radice dell'albero. Essa è misurata dal numero di classi "padre".

All'aumentare della profondità di una classe all'interno della gerarchia, aumenta il numero di metodi che è possibile ereditare, rendendo più complessa la predizione del suo comportamento.

Una più grande profondità dell'albero costituisce una maggiore complessità del progetto, in quanto sono chiamati più metodi e più classi, ma è maggiore il potenziale riuso dei metodi ereditati.

Questa metrica misura soprattutto l'efficienza e la possibilità di riuso, ma è anche riferita alla comprensibilità e alla collaudabilità.

Una metrica di supporto per DIT è il *Number Of Methods Inherited* (NMI: numero di metodi ereditati).

In riferimento alla figura 7 *Dipartimento vendite* è la radice dell'albero e ha DIT = 0; il DIT per la classe *Abbigliamento* è pari a 1.

NOC - Number of Children (Numero di Figli)

La metrica Number of Children è il numero di sottoclassi immediatamente subordinate ad una certa classe della gerarchia. Essa è un indicatore della potenziale influenza che una classe può avere sul progetto e sul sistema.

Maggiore è il valore di NOC, maggiore sarà la probabilità di avere un'astrazione errata dei "genitori" e può indicare un uso improprio delle sottoclassi.

Tuttavia, maggiore è il valore di NOC, maggiore sarà il riuso, in quanto l'eredità può essere considerata come una forma di riuso.

Se una classe ha un valore elevato di NOC, può richiedere più testing dei metodi di quella classe, e perciò cresce il tempo necessario per il testing.

Questa metrica valuta soprattutto l'efficienza, il riuso e la collaudabilità.

In riferimento alla figura 7 la classe *Dipartimento Vendite* ha NOC = 2 mentre le classe *Abbigliamento* ha NOC = 0 in quanto è un nodo foglia della struttura dell'albero.

Mark Lorenz e Jeff Kidd

Secondo l'approccio di Lorentz-Kidd le *Design Metrics* si dividono in 27 metriche raggruppate in 5 sotto categorie: *Method Size*, *Method Internals*, *Class Size*, *Class Inheritance*, e *Class Internals*. Per semplicità ci soffermeremo soltanto in alcune di queste metriche:

Method Size (Dimensione di Metodo)

Consiste in due misure: *Number of Message Sends* e *Lines of Code* (LOC).

La metrica *Number of Message Sends* può essere usata per conoscere "l'intensità del flusso informativo" fra le applicazioni. Questo è identificato da tre tipi di messaggi:

- *unary messages*: messaggi mandati senza argomenti.
- *binary messages*: messaggi mandati con un solo argomento e separati da speciali selettori (per esempio: concatenazione di stringhe o funzioni matematiche)
- *keyword messages*: messaggi mandati con uno o più argomenti.

Method Internals (Interiorità di Metodo)

Consiste in due misure: *Method Complexity* and *Strings of Message Sends*.

La metrica *Method Complexity* cerca di sostituire l'analoga misura (*Complessità Ciclomatica* di McCabe) relativa alla programmazione procedurale con una misura più adatta ai sistemi object-oriented.

La *Complessità Ciclomatica* di McCabe fu sviluppata quando erano ancora diffusi linguaggi come COBOL e FORTRAN, e le unità consistevano in diverse dozzine di linee di codice. La complessità non era facilmente determinabile e perciò aveva senso usare questo tipo di misura. Tuttavia se consideriamo un tipico codice object-oriented si può notare che esso è composto da tante unità logiche, ognuna delle quali contiene al massimo una dozzina di linee di codice (in media fra 5 e 10). Di conseguenza la complessità di una singola unità è di solito molto bassa.

9.2.3. Ulteriori metriche per sistemi object-oriented

Altre classificazioni prendono in esame

- **Metriche strutturali**
- **Metriche orientate al riuso**
- **Metriche orientate agli aspetti "umani"**
- **Metriche varie**

Metriche strutturali

Queste metriche riguardano le strutture più significative di un codice object-oriented come i metodi, gli oggetti, le variabili globali, etc. Lo scopo principale di queste metriche è quello di valutare le relazioni esistenti fra i vari blocchi di un sistema software object-oriented.

Si tratta di metriche che riguardano gli aspetti statistici del codice sorgente di un progetto object-oriented.

Number of Methods (numero di metodi)

Questa metrica potrebbe essere importante nel valutare la complessità di ogni oggetto, in quanto maggiore è il numero di metodi contenuti in un oggetto, più elevata è la complessità dell'oggetto stesso. Inoltre studiando e raccogliendo dati relativi all'utilizzo di questa metrica si può determinare la complessità ideale di un oggetto, con la possibilità di stabilirne uno standard di complessità.

Number of Objects (numero di oggetti)

Tale metrica fornice un'idea della complessità totale del sistema. Questa misura infatti potrebbe essere importante per determinare, dopo aver raccolto una quantità sufficiente di dati da vari progetti completati, una relazione tra la complessità del problema in esame e il numero di oggetti.

Number of Messages (Numero di Messaggi)

Contando il numero di messaggi che un oggetto invia, si può quantificare il suo grado di comunicazione con l'esterno, indice del grado di accoppiamento fra oggetti.

Se un oggetto può rispondere ad una grande quantità di messaggi, inviati da altrettanti oggetti, significa che esiste un elevato grado di accoppiamento, cosa non favorevole al riuso del codice, al testing e al debugging.

Number of Receiving Objects (Servers) (Numero di oggetti che ricevono)

Maggiore il numero di oggetti che possono ricevere messaggi (oggetti *Server*) in un sistema, più probabile è che molti di questi stiano eseguendo pochissime funzioni. Questo potrebbe essere un indizio che il sistema è stato progettato utilizzando troppi oggetti, cosa generalmente sfavorevole.

Number of Sender Objects (Actors) (Numero di oggetti che inviano)

Problema simmetrico al precedente. Maggiore il numero di oggetti che possono inviare messaggi (oggetti *Actor*) in un sistema, più probabile è che molti di questi stiano eseguendo pochissime funzioni. Il rapporto tra gli *Actor* e i *Server* dipende anche dal tipo di applicazione.

Number of Agent Objects (Numero di oggetti Agent)

Gli oggetti *Agent* hanno sia le proprietà degli *Actor* che dei *Server*; perciò se un sistema contiene un gran numero di *Agent*, allora ci potrebbero essere molti messaggi inviati tra i vari oggetti. Questo porta ad avere un sistema altamente accoppiato riducendo drasticamente la possibilità di riuso di un codice. Potrebbe inoltre essere difficile la correzione di eventuali errori di sistema.

Number of Global Variables in each Class (numero di variabili globali in ogni classe)

Questa metrica è utile per determinare le relazioni di eredità fra i diversi oggetti.
Un grande numero di variabili globali significa che gli oggetti in posizioni basse della gerarchia possono ereditare molte caratteristiche che si trovano in posizioni più alte. Tuttavia, molte variabili globali possono diminuire l'indipendenza di un oggetto.

Number of Levels in the Class Hierarchy Tree (Numero di livelli nell'albero gerarchico della classe)

Dalla misura della profondità dell'albero gerarchico, si è in grado di conoscere il livello di astrazione delle classi. Questa metrica può essere utile nel decidere il livello di specializzazione opportuno e la dipendenza fra oggetti simili in un sistema.

Number of Leaves in the Class Hierarchy Tree (Numero di foglie nell'albero gerarchico della classe)

Misurare l'ampiezza di un albero gerarchico della classe è importante quanto conoscerne la profondità. Questa metrica può mettere in evidenza le relazioni di eredità fra le superclassi e le sottoclassi; può essere inoltre un indice della complessità totale del sistema.

Ratio Between Depth and Breadth (Rapporto tra profondità e ampiezza)

Tale rapporto potrebbe essere più importante che misurare profondità e ampiezza separatamente.

Ratio of Methods/Class (Rapporto Metodi/Classe)

Il numero di metodi implementati in una classe dà una misura della sua complessità.

Ratio of private/public methods (Rapporto fra metodi privati/pubblici)

Tale metrica è utile per determinare il grado di inversa proporzionalità fra l'astrazione dei dati e l'eredità. Questo può essere adattato all'utilizzo di vari linguaggi di programmazione object-oriented in varie applicazioni.

Ratio of Abstract/ Instantiated Objects (Rapporto fra oggetti astratti e istanziati)

Da tale rapporto ci si può fare un'idea sulle relazioni gerarchiche all'interno del sistema. Per esempio, un alto valore di tale rapporto può indicare una struttura gerarchica con alta eredità.

Ratio of lines of code/method (Rapporto fra linee di codice e metodi)

Questa metrica fornisce un'indicazione sulla complessità di ogni metodo. Il problema di come determinare il suo valore ottimale per un certo linguaggio è ancora aperto.

Ratio of lines of code/comment (Rapporto fra linee di codice e di commento)

Questo rapporto dipende molto dal programmatore. Sebbene sia buona norma di programmazione documentare opportunamente tutti gli aspetti del sistema in modo da facilitare il riuso e la modifica del codice, un numero troppo elevato di commenti può comportare una perdita di efficienza.

Metriche orientate al riuso

Questo gruppo di metriche focalizza la sua attenzione sul riuso. Il riuso del codice è uno degli attributi più ricercati nell'approccio object-oriented, in quanto rende versatile ed efficiente un sistema.

Si è dimostrato che il software prodotto con oggetti "riusati" è più affidabile: gli oggetti "riusati", infatti, a differenza di quelli appena creati, sono già stati testati a sufficienza, obbligando a una minore necessità di manutenzione.

Due importanti metriche rientrano in questo gruppo:

Number of Objects Reused (Numero di oggetti riusati)

Tale metrica consiste semplicemente nel tener conto del numero di oggetti non modificati e pre-codificati che sono stati utilizzati dal sistema. In generale, quanto più è alto il numero di oggetti riusati, tanto più è probabile che il sistema sia complessivamente più affidabile.

Percent of Reused Objects Modified (percentuale di oggetti riusati che vengono modificati)

Una volta che un oggetto riusato viene modificato, per esso non sarà più garantita l'affidabilità come nel caso precedente. Perciò non è più sufficiente conoscere il numero di oggetti riusati, ma anche il numero di oggetti riusati che sono stati successivamente modificati; ciò comporta che dopo ogni modifica un oggetto riusato debba essere ulteriormente testato prima della sua implementazione.

Metriche orientate agli aspetti "umani"

Appartengono a questo gruppo quelle metriche che riguardano gli aspetti "umani" della programmazione object-oriented, spesso trascurati dall'analisi oggettiva dei tipi precedenti di metriche. Essendo tuttavia molto difficile misurare tale tipo di fattori, si cerca di tenerne conto tramite l'intuito e l'esperienza.

Metriche varie

Method Hiding Factor (MHF)

Definiamo l'invisibilità di un metodo come la percentuale di classi totali dalle quali tale metodo non sia visibile. Da notare che i metodi ereditati non vengono considerati.

La metrica MHF è definita come il rapporto fra la somma delle invisibilità di tutti i metodi definiti in tutte le classi e il numero totale di metodi definiti nel sistema preso in considerazione.

Attribute Hiding Factor (AHF)

La metrica AHF è definita come il rapporto della somma delle invisibilità di tutti gli attributi definiti in tutte le classi e il numero totale di attributi definiti nel sistema preso in considerazione.

Method Inheritance Factor (MIF)

La metrica MIF è definita come il rapporto fra la somma dei metodi ereditati di tutte le classi del sistema preso in considerazione e il numero totale di metodi disponibili (quelli definiti localmente più quelli ereditati) per tutte le classi.

Attribute Inheritance Factor (AIF)

La metrica AIF è definita come il rapporto fra la somma degli attributi ereditati di tutte le classi del sistema preso in considerazione e il numero totale di attributi disponibili (quelli definiti localmente più quelli ereditati) per tutte le classi.

Polymorphism Factor (PF)

La metrica PF è definita come il rapporto fra il numero attuale di possibili differenti situazioni di polimorfismo per la classe C_i e il numero massimo di possibili distinte situazioni di polimorfismo per la stessa classe C_i.

Coupling Factor (CF)

La metrica CF è definita come il rapporto fra il numero massimo possibile di accoppiamenti nel sistema e il numero attuale di accoppiamenti non imputabile all'eredità.

Internal Privacy (INTP)

La metrica INTP si riferisce all'uso di funzioni accessorie anche all'interno di una classe.

Category Naming (CATN)

La metrica CANT divide le classi in insiemi semanticamente significativi.

9.3. Linee direttive di interpretazione

Vediamo ora come possono essere interpretati i valori delle metriche per migliorare la qualità e l'efficienza dei sistemi object-oriented.

Esistono diversi modi per interpretare tali valori, in particolare il SATC (*Software Assurance Technology Center* presso la *NASA*) ha proposto delle linee direttive di interpretazione basate sul confronto dei valori delle metriche fra moduli usati come test e i moduli in esame. In generale, il fatto che esista una differenza fra tali valori non significa che il modulo in esame non sia efficiente, ma casomai che è necessaria un'analisi più approfondita.

Nella tabella seguente sono riportate delle indicazioni sui valori ottimali per alcune delle metriche trattate:

METRICHE	Valore obiettivo
Cyclomatic Complexity (CC)	Basso
Lines of Code/Executable Statements (LOC/EXEC)	Basso
Comment Percentage (CP)	~ 20 – 30 %
Weighted Methods per Class (WMC)	Basso
Response For a Class (RFC)	Basso
Lack Of Cohesion of Methods (LOCM)	Basso/Alto
Coupling Between Objects (CBO)	Basso
Depth of Inheritance (DIT)	Basso (trade-off)
Number of Children (NOC)	Basso (trade-off)

Tabella 15: Linee direttive di interpretazione

Tuttavia, come indicato nelle ultime due metriche, esiste una relazione di inversa proporzionalità (trade-off) fra diverse metriche: un alto valore di DIT può aumentare la complessità della manutenzione, ma comporta anche un miglioramento del riuso; così come un alto valore di NOC può aumentare i tentativi di testing ma facilita anch'esso il riuso.

Un progettista deve essere a conoscenza delle relazioni fra le strutture e del fatto che, alterando i valori di una certa metrica, si possono avere effetti sul testing, la comprensibilità, la manutenzione e il riuso.

9.4. Rappresentazione e interpretazione grafica delle metriche

In questo paragrafo vedremo come è possibile rappresentare e interpretare graficamente i valori assunti dalle metriche (in particolare le metriche CK). Per alcune metriche è sufficiente un semplice istogramma per rappresentare i valori prevalenti ed estremi.

Per esempio in figura 10 è mostrato un istogramma che rappresenta per ogni valore di WMC quante classi del progetto hanno tale valore. Come si vede, mentre molte classi hanno WMC<20, ci sono alcune classi con WMC>100. Per queste ultime è necessaria un'ulteriore controllo e/o modifica.

Questo istogramma è utile anche per monitorare la complessità totale.

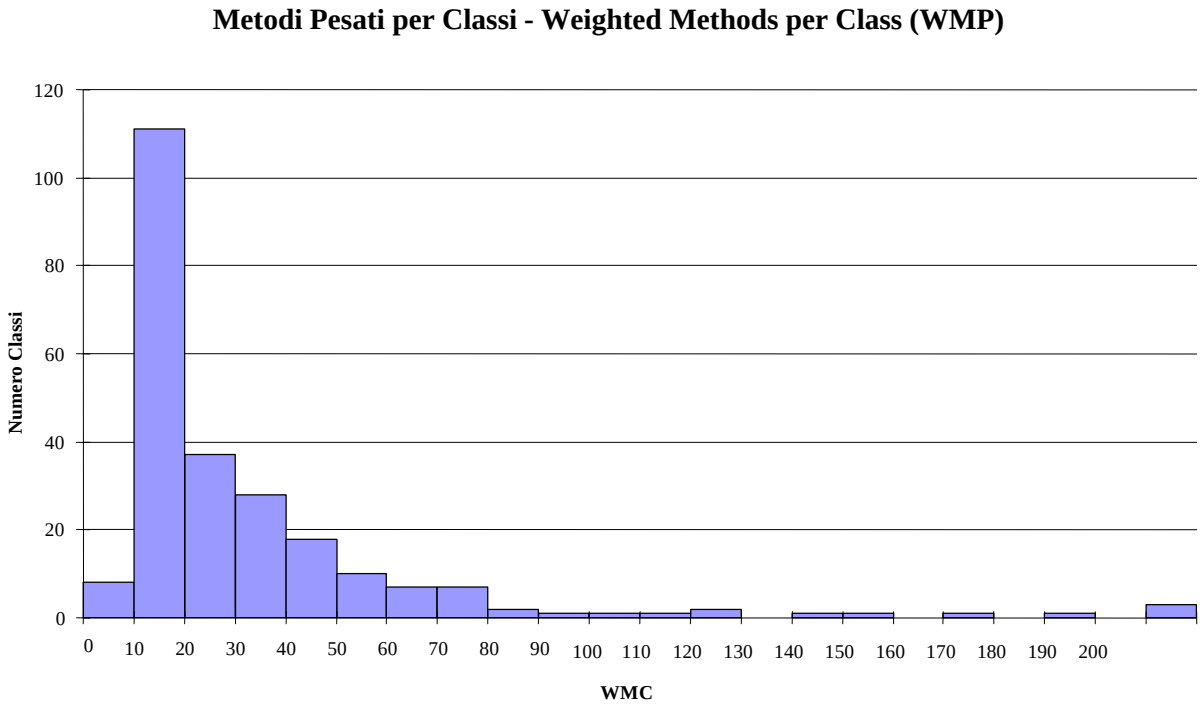


Figura 10: Istogramma dei valori di WMC

E' possibile tracciare istogrammi simili anche per altre metriche.

L'istogramma in figura 11, riferito alla metrica RFC, rappresenta un progetto in cui alcune classi sono capaci di chiamare più di 200 metodi. Classi con un così grande valore di RFC hanno un'elevata complessità e una bassa comprensibilità e perciò il testing e il debugging saranno più complicati.

Anche questo istogramma può essere utile per monitorare la complessità totale.

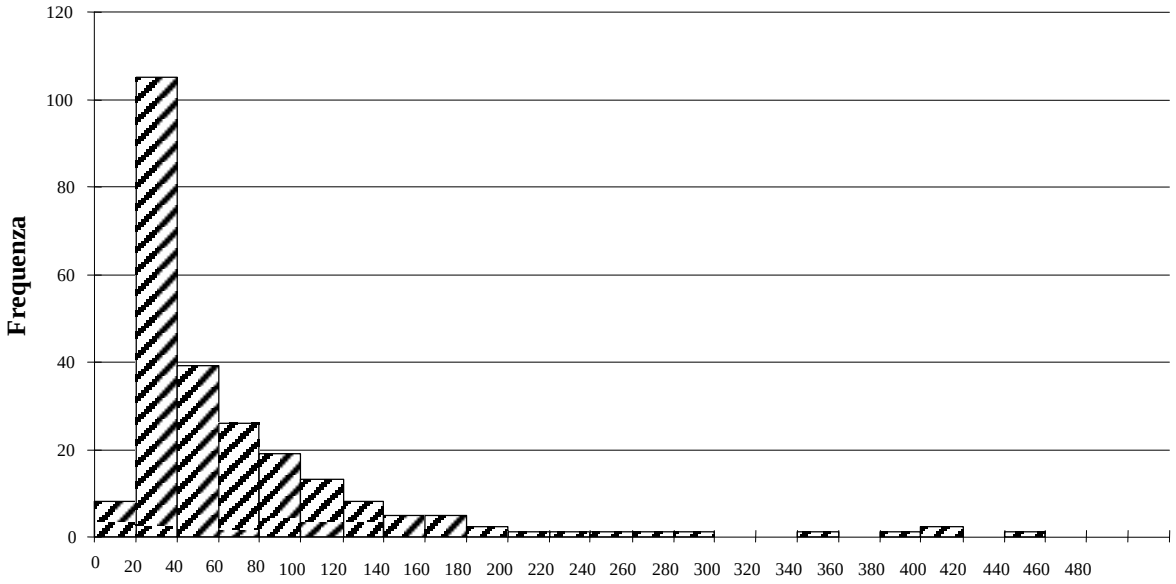


Figura 11: Istogramma dei valori RFC

In figura 11 è mostrato un istogramma relativo alla metrica CBO. Delle 240 classi in questo progetto, più di un terzo sono indipendenti in quanto hanno un valore CBO=0. Valori alti di CBO indicano classi poco comprensibili, improbabili da riutilizzare e di difficile manutenzione.

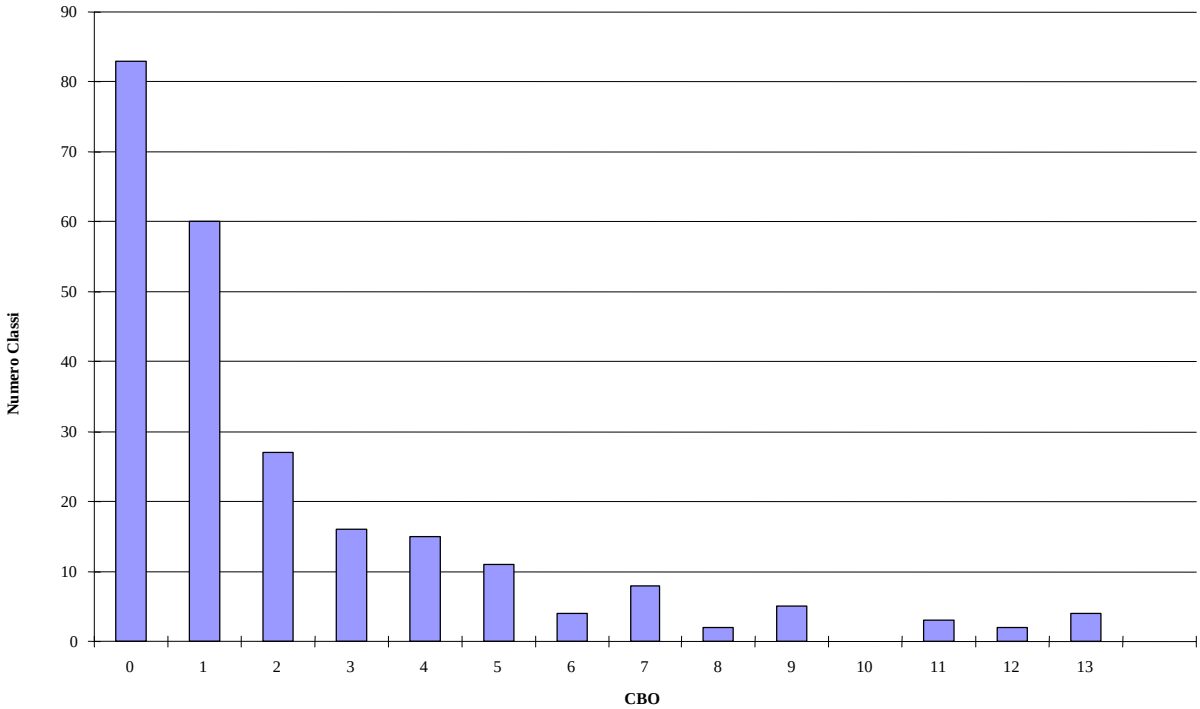


Figura 12: Istogramma dei valori CBO

Le metriche riferite alla struttura gerarchica (NOC e DIT) possono essere rappresentate graficamente come mostrato in figura 12.

Come abbiamo visto, una classe con DIT=0 è la radice dell'albero. Se essa è anche una foglia (NOC=0), allora si tratta di un unico codice che non porta benefici per l'eredità e il riuso.

Nel progetto rappresentato in figura 10, quasi il 66% delle classi si trovano al di sotto di altre classi dell'albero, il che indica un moderato livello di riuso. Percentuali più alte di 2 - 3% per la metrica DIT mostrerebbero un alto grado di riuso, ma anche una crescente complessità.

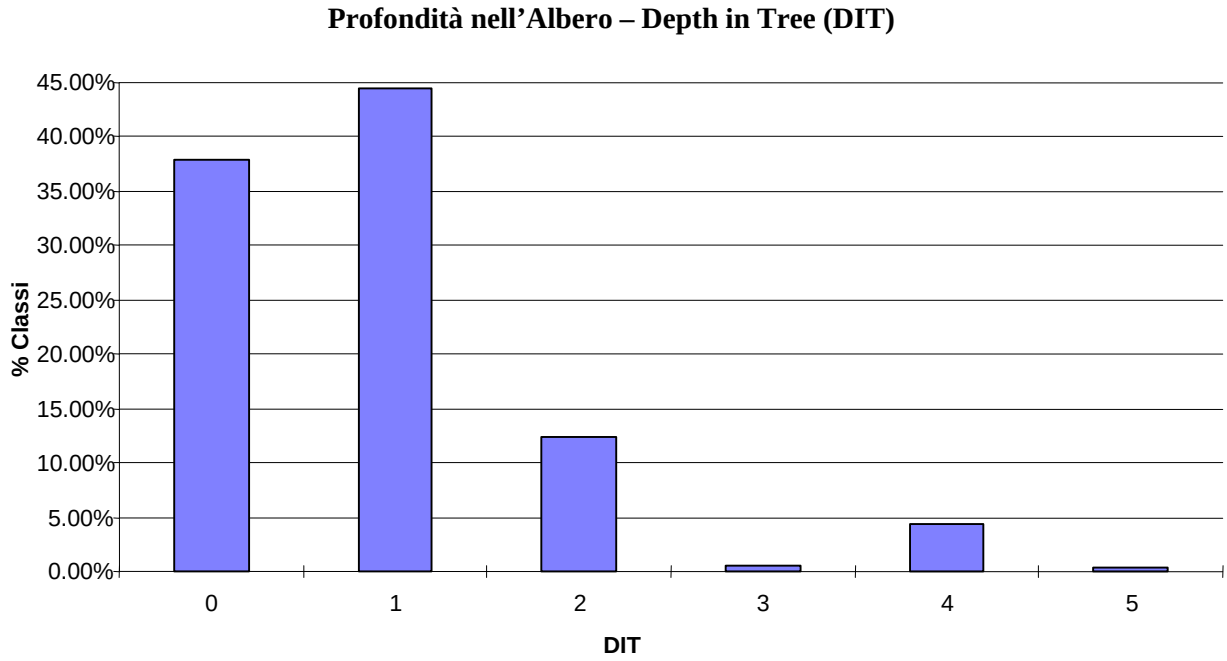


Figura 13: Istogramma dei valori DIT

Il valore della metrica LOCM dipende dal numero dei metodi, perciò esisterà un valore massimo possibile. In figura 13 è mostrato un grafico di valori LCOM paragonati al loro massimo possibile.

L'esperienza dice che il valore migliore di LCOM per un certo progetto è il più piccolo ottenuto dal confronto con il suo massimo.

A partire dal grafico viene identificato il valore più vicino all'asse osservando i valori di LCOM. Anche il SATC usa la linea di tendenza mostrata nel grafico per fare confronti tra progetti e tra linguaggi.

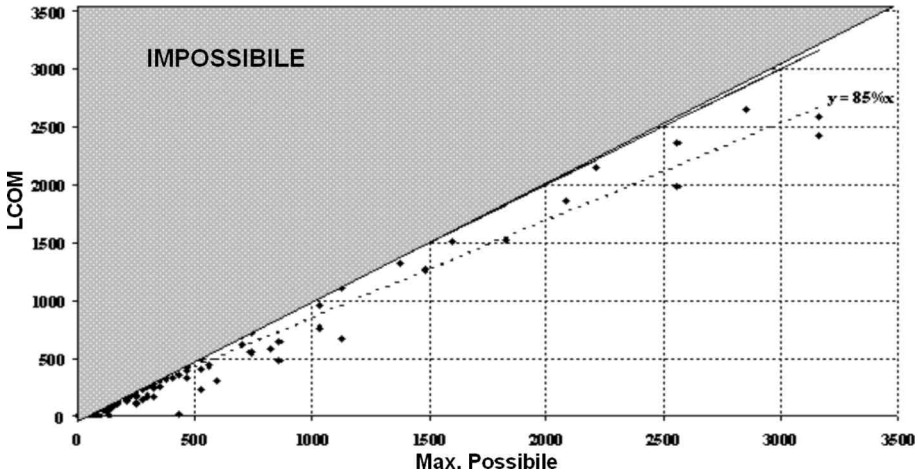
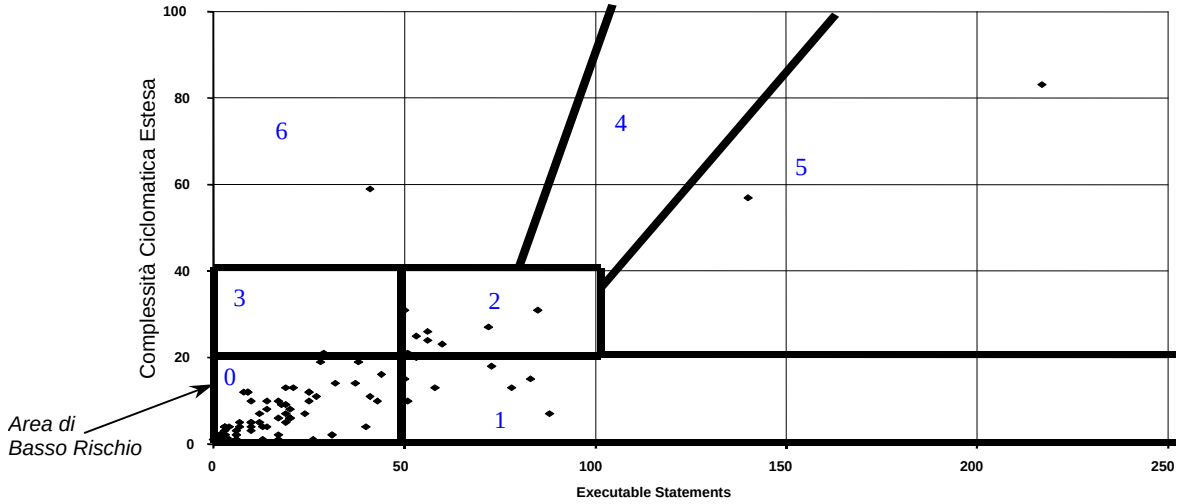


Figura 14: Valori di LCOM

Per molte metriche, è più efficace analizzare i moduli utilizzando due metriche. In figura 14 i metodi sono rappresentati basandosi sulla dimensione e sulla complessità.

Il SATC ha fatto delle ricerche per identificare i valori di riferimento e le cosiddette "regioni a rischio" che rappresentano le zone in cui i metodi hanno potenzialmente una cattiva qualità che influenzerà la manutenzione, il riuso e la comprensibilità. Queste regioni a rischio, elaborate inizialmente per codici non object-oriented, sono state pensate per diminuire in numero con ulteriori ricerche.

Figura 15: Confronto fra dimensione e complessità



La seguente tabella riassume il diagramma in figura 15.

	Rischio 0	Rischio 1	Rischio 2	Rischio 3	Rischio 4	Rischio 5	Rischio 6	Totale
Conto	85	7	9	1	0	2	1	105
Percentuale	81.0%	6.7%	8.6%	1.0%	0.0%	1.9%	1.0%	100.0%
				Rischio elevato a causa della dimensione e della complessità				

Tabella 16: Confronto fra dimensione e complessità

In figura 16, la metrica RFC è rappresentata in relazione al numero di metodi. I punti sopra o vicino alla linea di demarcazione tra la zona “possibile” e la zona “non possibile”, rappresentano classi che non chiamano metodi esterni. Questo indica ai progettisti che ci sono alcune classi con più di 40 metodi che possono influire anche su molti oggetti in altre classi.

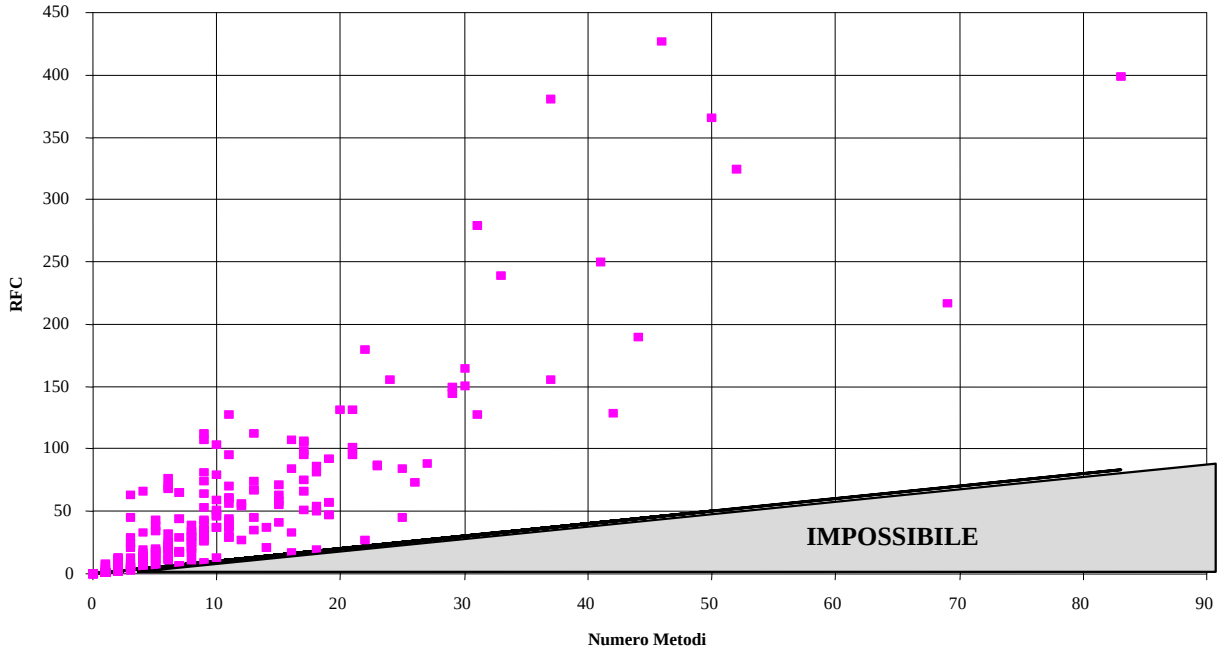


Figura 16: Risposta per Classe (RFC) /Numero di Metodi

Come già discusso, quando si cerca di determinare il valore appropriato delle metriche NOC e DIT, esiste un rapporto di inversa proporzionalità: un valore alto di DIT indica l’inversa proporzionalità fra la crescente complessità e il decrescente riuso; analogamente valori alti di NOC indicano un migliore riuso, ma potrebbero richiedere un maggiore testing.

In base a tale considerazione, la figura 17 mostra che esiste una classe di interesse (l’unica che si trova a tre passi dalla radice e ha 40 figli), che rappresenta il giusto compromesso tra le due metriche.

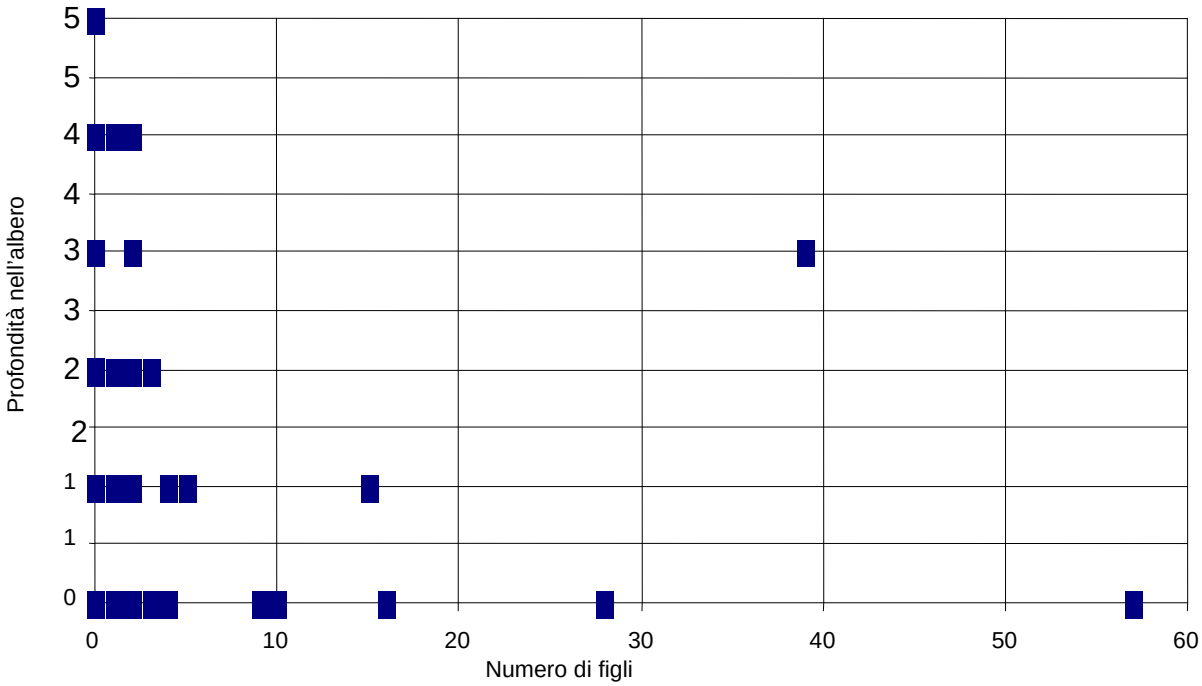


Figura 17: Valutazione della gerarchia

Riassumendo possiamo dire che le metriche object-oriented aiutano a valutare lo sviluppo e il testing necessari per la comprensibilità, la manutenzione e il riuso. Questo è riassunto nella seguente tabella:

Metrica	Obbiettivo	Testing	Comprensibilità	Manutenibilità	Sviluppo	Riuso
Complessità	↓	↓	↑	↑		
LOC	↓	↓	↑	↑		
Commenti %	↑	↓	↑	↑	↓	
WMC	↓			↑	↓	↑
RFC	↓	↓				↑
LCOM	↓		↑	↑	↓	↑
CBO	↓	↓	↑	↑		↑

Tabella 17: Tabella riassuntiva per le metriche object-oriented

10. Applicazione

10.1. Esempio di un'applicazione finanziaria

In questo capitolo vedremo un esempio di applicazione che utilizza le metriche CK definite nel capitolo precedente.

Consideriamo un possibile sistema object-oriented (OO) chiamato TRADER, concepito per fornire un valido aiuto ad un operatore (trader) di una società durante le transazioni finanziarie. Tale sistema fornisce, attraverso un'interfaccia grafica (GUI), degli strumenti utili per raccogliere i particolari di una transazione corrente.

TRADER è in grado di stampare le descrizioni di una transazione unica e di raccogliere in un database tutte le attività commerciali intraprese. Inoltre ha la capacità di comunicare con altri sistemi per reperire informazioni pertinenti (per esempio le quote dei titoli in borsa e i tassi di scambio).

Tale esempio è la versione altamente semplificata di uno dei sistemi reali: il sistema "TPM", che verrà analizzato successivamente.

La figura 18 mostra la struttura di classe, in pseudo codice, per tutte le quattro classi del sistema; mentre la figura 19 più sotto mostra la gerarchia di classe del sistema. Si noti che in entrambe le figure, le chiamate di metodo sono indicate con la parola chiave "call" e che gli accessi alle variabili (instance) sono indicate nei commenti.

E' presente una classe "radice" detta *trade* e tre sottoclassi denominate *equity trade*, *bond trade* e *foreign exchange* (*fx) trade* che ereditano dalla "radice" gli stessi metodi e le stesse variabili.

```
// class trade
Attributes                              // a.k.a. instance variables
trade id, counterparty, trade value     // details of the trade
Operations                              // a.k.a. methods
call evaluate_counterparty()            // determines validity of the counterparty
                                        // from a data base file
call get_trade_id()                     // obtain trade id details from the user
call position_update ()
call position_manager::report_trade()// send message to external
                                        // position_manager class furnishing trade
                                        // value and trade id.
----------------------------
// class bond trade
Attributes                              // a.k.a. instance variables
bond_details                            // bond ratings details
Operations                              // a.k.a. methods
call get_bond_info()                    // access bond data base for current market
                                        // price information
----------------------------
// class fx trade
Attributes                              // a.k.a. instance variables
forex_details                           // foreign market information
Operations                              // a.k.a. methods
call calculate_exchange_rates()         // access currency market monitor for
                                        // latest rates
----------------------------
// class equity trade
Attributes                              // a.k.a. instance variables
company, stock_market, PE ratio, earnings, 52_week_hi&lo
Operations                              // a.k.a. methods
call estimate_beta()                    // determine risk compared to rest of the
                                        // market
call get_stock_quotes()                 // consult external stock quotation data
                                        // base
 call quotron::quotes() for trade_id // get the latest quote for the trade
```

Figura 18: Pseudo codice per le classi in TRADER

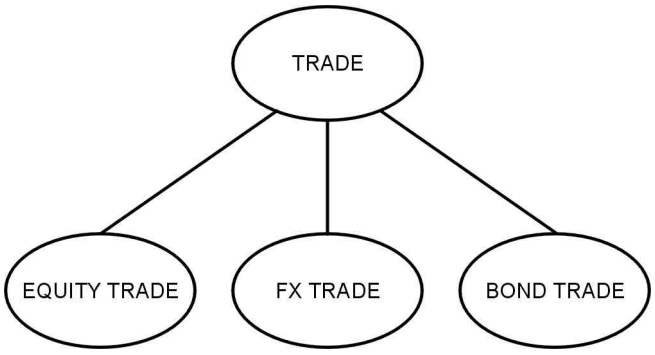


Figura 19: Gerarchia di classe per il possibile TRADER system

Prima di valutare i valori delle sei metriche CK per questo esempio, richiamiamo brevemente la loro definizione:

NOC è definita come il numero delle sottoclassi presenti;

WMC è il numero di metodi in una classe (assumendo pesi unitari per tutti i metodi);

DIT è la lunghezza del cammino massimo dalla radice della gerarchia di classe;

CBO è approssimativamente il numero di coppie con altre classi (dove per coppia si intende la chiamata di un metodo o di una variabile da un'altra classe);

RFC è la cardinalità dell'insieme di tutti i metodi che possono rispondere a un messaggio diretto verso un oggetto di una classe;

LCOM è il numero delle coppie di metodi che non hanno in comune variabili meno quelle che invece le hanno.

Tornando al nostro esempio, la classe *trade* ha tre sottoclassi (*equity trade*, *fx trade* e *bond trade*), tre metodi (*evaluate_counterparty*, *get_trade_id* e *position_update*), ed è una classe radice. Quindi per tale classe il valore delle metriche NOC, WMC, DIT è rispettivamente 3, 3 e 0.

I valori di NOC, WMC e DIT per le altre classi sono riportate in tabella; notare che tali classi non hanno figli (NOC=0) e hanno un solo "padre" (DIT=1).

La metrica CBO vale 1 per la classe *trade* poiché essa utilizza un metodo esterno (*report_trade*) prelevato da un'altra classe (*position_manager*); analogamente CBO vale 2 per la classe *equity trade* poiché essa utilizza una variabile (*trade id*) un metodo (*quotes*) esterni da altrettante classi (*trade* e *quotron*). Per le altre classi CBO è zero perché non sono accoppiate con altre classi.

La metrica RFC vale 4 per la classe *trade* poiché l'arrivo di un messaggio a questa classe potrebbe lanciare l'esecuzione di 4 metodi (tre dichiarati in *trade* e il *report_trade* chiamato dal *position_manager*). Analogamente per la classe *equity trade* RFC vale tre perché essa può lanciare l'esecuzione di tre metodi (due dichiarati in *equity trade*, uno prelevato da *quotron*).

In *trader* ci sono tre metodi e perciò in totale ci sono tre coppie di metodi:

evaluate_counterparty & get_trade_id,

evaluate_counterparty & position_update

get_trade_id & position_update

La prima e la seconda coppia non hanno nessuna variabile in comune; mentre la terza coppia ne ha una (*trade id*). Di conseguenza il numero di coppie di metodi disgiunti è pari a 2, il numero di metodi simili è pari a 1, e quindi la metrica LCOM è pari a 2–1=1.

Analogamente per la altre classi LCOM vale zero. In particolare per *equity trade* si è assunto che *estimate_beta* e *get_stock_quotes* usino entrambi una variabile comune.

Tutte le sei metriche per le quattro classi sono mostrate nella tabella seguente:

Nome Classe	WMC	DIT	NOC	CBO	RFC	LCOM
trade	3	0	3	1	4	1
bond trade	1	1	0	0	1	0
fx trade	1	1	0	0	1	0
equity trade	2	1	0	2	3	0

Tabella 18: Metriche CK per le classi di TRADER

Adesso esaminiamo un sistema più complesso, nominata TRADER Versione 2.0, dove le operazione riguardanti il commercio delle obbligazioni comprendono sia quelle pubbliche che quelle private. Le obbligazioni pubbliche hanno un differente tasso d'interesse e una differente previsione di scadenza rispetto a quelle private, così la gerarchia di classe potrebbe essere cambiata in modo da utilizzare *municipal bond trade* e *corporate bond trade* come particolarizzazioni della classe *bond trade*; ogni sottoclasse può avere operazioni di calcolo separate, tuttavia dovrà separare gli attributi e le operazioni che sono dichiarate nella classe *bond trade*.

Inoltre TRADER si preoccupa sia delle equità nazionali che internazionali. Possono essere dichiarate due nuove sottoclassi per specificare la classe *equity trade*. Le dichiarazioni delle classi e la nuova gerarchia sono mostrate nelle figure 19 e 20.

I valori delle metriche per le classi del nuovo sistema sono mostrate nella seguente tabella:

Nome Classe	WMC	DIT	NOC	CBO	RFC	LCOM
trade	3	0	3	1	4	1
bond trade	1	1	2	0	1	0
fx trade	1	1	0	0	1	0
equity trade	2	1	2	2	3	0
municipal bond trade	1	2	0	1	2	0
corporate bond trade	1	2	0	1	2	0
international equity	2	2	0	1	3	1
domestic equity	0	2	0	0	0	0

Tabella 19: Metriche CK per le classi di TRADER (Version 2.0)

```
// class municipal bond trade
Attributes                    // a.k.a. instance variables
State_or_federal, over_the_counter
Operations                    // a.k.a. methods
Calculate_coupon_rate()       // determine the inter-rate rate for the bond
  call Tbill_server::rates() // get the current rates from another source
----------------------------
// class corporate bond trade
Attributes                    // a.k.a. instance variables
adr, sp_rating
Operations                    // a.k.a. methods
calc_rating(sp_rating)        // get the standard & poors rating
      if adr == TRUE then call fx trade::calculate_exchange_rates()
                              // get the exchange rate information if this is a
                              // foreign bond issue
----------------------------
// class international equity
Attributes                    // a.k.a. instance variables
exchange_rate, quotation
Operations                    // a.k.a. methods
perform_analysis_roa (fx trade::calculate_exchange_rates())
                              // analyze the stock using the foreign exchange
                              // rate
get_quotron(quotation)        // determine the current price of stock
----------------------------
// class domestic equity
Attributes                    // a.k.a. instance variables
attribute1
Operations                    // a.k.a. methods
none defined yet
```

Figura 20: Pseudo codice per le classi in TRADER (Version 2.0)

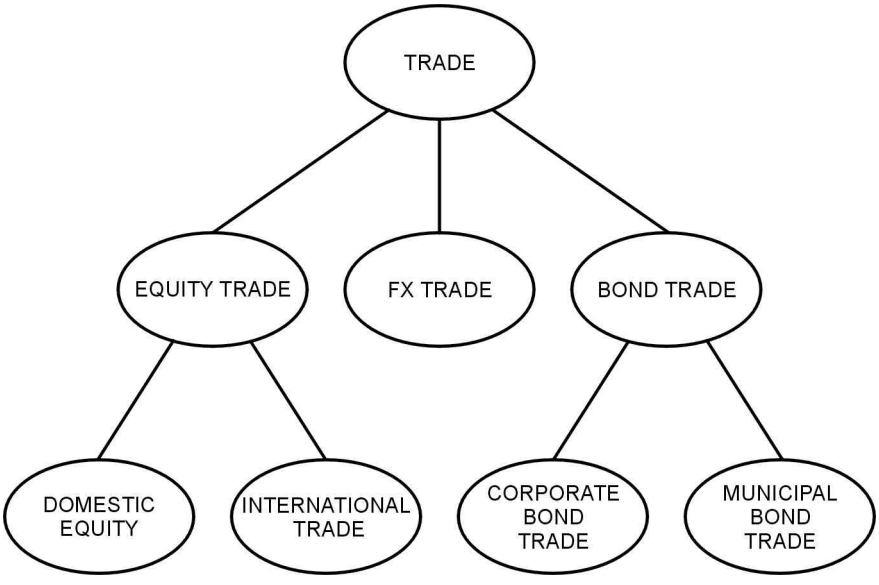


Figura 21: Gerarchia di classe per il possibile TRADER system (Versione 2)

Si noti come le metriche CBO e RFC per le classi *corporate bond trade* e *international equity* hanno valori più grandi, poiché per avere informazioni sul tasso di cambio dipendono dalla classe *fx trade*. Un attento progettista avrà sicuramente notato che questa informazione non è pertinente solo per il commercio del cambio estero, e sposterà il metodo del calcolo del tasso alla classe *trade*, in maniera tale che tutti i tipi di commerci possano avere facile accesso a tale classe. Tale scorciatoia ha avuto l'effetto di accoppiare due classi "foglia" dell'albero.

Se le metriche CK fossero riportate durante una revisione del progetto o in un rapporto di analisi del sistema, l'incremento nei valori delle metriche CBO, RFC e WMC per le classi foglia del sistema come *international equity* e *corporate bond* sta ad indicare che forse l'incapsulamento è stato compromesso e le opportunità per il riuso attraverso l'eredità sono state perse. In tal modo i compromessi nelle norme di progetto possono condurre a classi che sono più difficili da capire, testare e riutilizzare in altre applicazioni.

Si noti anche come la classe di equità nazionale, mostrata nella tabella 3, ha in prevalenza zeri; come si può vedere dal pseudo codice, ciò è dovuto dal fatto che tale classe rimane in gran parte non specificata. Ciò è possibile (purché non ci siano troppi zeri) e permette ad un progettista di trarre utili informazioni circa quelle classi che non sono specificate in maniera appropriata; così come è importante porre attenzione a valori elevati delle metriche.

Ricapitolando, questo esempio mostra:

come le metriche vengono calcolate per le singole classi;

che le metriche vengono modificate con gli sviluppi del progetto;

che anche un sistema elementare può velocemente produrre una inutile complessità che lede l'integrità dell'architettura del progetto;

che le metriche possono essere utilizzare durante la fase di progetto per ridurre e controllare la complessità.

A. Appendice: Richiami alle strutture object-oriented

A.1. Definizioni Generali

- **Classe:** è l'implementazione di una struttura astratta che esporta all'esterno soltanto alcune operazioni. Una classe consiste sia in un modello (*pattern*) che in un meccanismo per creare degli elementi (*items*) basati sul modello della classe. Tali elementi sono dette **istanze**. Secondo questo punto di vista una classe può essere considerata sia come una "fabbrica di istanze", che come un'astrazione delle sue stesse istanze.
- **Metaclasse:** è una classe le cui istanze sono loro stesse delle classi. Alcuni linguaggi di programmazione object-oriented supportano direttamente delle metaclassi definite dall'utente. In effetti le metaclassi possono essere viste come classi di classi, cioè per creare un'istanza, noi forniamo dei parametri specifici alla metaclasse e questi sono usati per creare una classe.
- **Classe Parametrizzata:** è una classe i cui elementi possono essere parametrizzati tutti o in parte. Una nuova classe (utilizzabile immediatamente) può essere generata attraverso un operazione di *instantiating* di classe parametrizzata con i suoi parametri richiesti. Esempi di classi parametrizzate sono i modelli (templates) in C++ e le classi generiche in Eiffel.
- **Instantiation:** particolare operazione con cui è possibile creare una istanza della classe con conseguente legame o aggiunta di dati specifici. Tale istanza è un **oggetto** della classe che contiene in sé sia uno stato interno (dati) che le operazioni di lettura o di modifica dello stato stesso.
- **Metodi:** sono le componenti di un oggetto che rappresentano le operazioni che l'oggetto stesso può effettuare sui dati in risposta ad un messaggio. Essi sono definiti come parte della dichiarazione di una classe.
- **Messaggio:** è una richiesta di esecuzione di un operazione (metodo) mandata da un oggetto ad un altro oggetto.
- **Eredità (Inheritance):** è una relazione gerarchica fra classi, in cui una classe (subclasse) acquisisce le caratteristiche di una o più altre classi (superclassi). L'eredità è un'astrazione che diminuisce la complessità, riducendo il numero di operazioni e operatori, e facilita il riuso ma può rendere difficoltosa la progettazione e la manutenzione.
- **Attributo (attribute):** si tratta generalmente di un nome che definisce le proprietà strutturali delle classi, uniche all'interno della classe stessa.
- **Operazione (operation):** è un azione effettuata da o su un oggetto, disponibile a tutte le istanze della classe, non necessariamente unica.
- **Coesione (cohesion):** è il grado di relazione che lega un particolare metodo con un altro metodo della classe. La progettazione object-oriented tende a massimizzare la coesione in quanto la coesione favorisce l'incapsulamento.
- **Accoppiamento (coupling):** è una misura della forza di associazione che lega un'entità ad un'altra. Le classi (e gli oggetti) possono essere accoppiati in tre modi:
    - Due oggetti sono accoppiati se e solo se un oggetto manda un messaggio all'altro oggetto.

  - Due classi sono accoppiate quando i metodi dichiarati in una classe usano metodi o attributi di un'altra classe.
  - L'eredità introduce un accoppiamento significativo e saldamente fissato fra superclassi e le loro subclassi.
- **Polimorfismo**: Possibilità da parte di un'operazione di essere definita ed eseguita con metodi diversi opportunamente selezionati in base alle caratteristiche dei dati sui quali tali metodi devono operare.

A.2. Esempio 1

In riferimento alla figura sottostante possiamo vedere che un sistema object-oriented inizia con la definizione di una classe (classe A) che contiene attributi e metodi.

Una classe *figlio* (*child class*) eredita tutti gli attributi e le operazioni dalla classe *padre* o *superclasse*, i quali vanno ad aggiungersi ai propri attributi e operazioni. A sua volta, una classe *figlio* può diventare *padre* di altre classi formando un altro ramo della struttura gerarchica ad albero.

Nell'esempio le classi A1, A2 e A3 sono *figli* della classe A (*padre*) dalla quale ereditano tutti gli attributi e le operazioni che vanno ad aggiungersi agli attributi e operazioni specifiche delle classi A1, A2 e A3.

Gli oggetti B1 e B2 sono creati a partire rispettivamente dalle classi A1 e A2. Tali oggetti sono anche accoppiati in quanto interagiscono fra loro attraverso un messaggio.

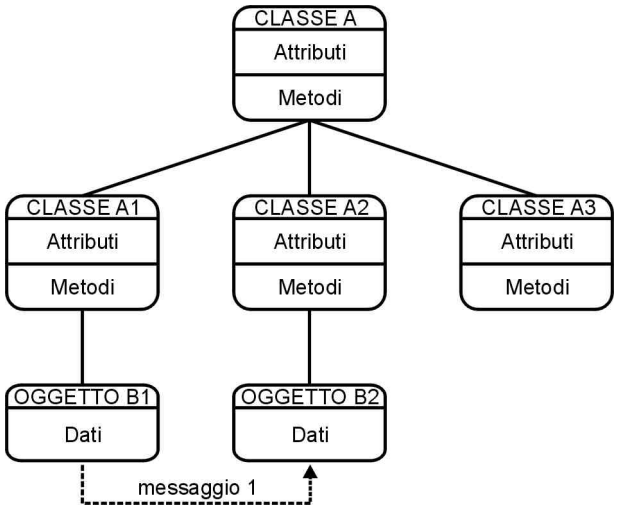


Figura 22: Esempio di gerarchia di classi

A.3. Esempio 2

Nella figura successiva è mostrato un esempio di un'applicazione con tre classi: la classe principale è *Dipartimento Vendite* che descrive un reparto di immagazzinamento generico. Questa classe ha due classi *figli*: *Abbigliamento* e *Elettrodomestici*.

Ogni reparto (cioè ogni oggetto della classe *Dipartimento Vendite*) avrà come attributi: *Direttore*, *Impiegati* (numero impiegati) e *Superficie Espositiva* che verranno ereditati dalle classi *figlio* e che vanno ad aggiungersi agli attributi *Sesso, Misura* e *Articolo* per la sottoclasse *Abbigliamento* e *Categoria* per la sottoclasse *Elettrodomestici*.

Gli oggetti sono *Abiti da Uomo* e *Abiti da Donna* per la classe *Abbigliamento* e *Grandi Elettrodomestici*, *Cucine* e *Elettronica* per la classe *Elettrodomestici*.

I metodi degli oggetti rappresentano le operazioni che il *Dipartimento Vendite* può effettuare. Esse sono *Mostra Merce, Vendita, Cambio Merce.*

Le classi *figlio* ereditano tutti questi metodi che vanno ad aggiungersi agli ulteriori metodi *Camerino* per la classe *Abbigliamento* e *Consegna e Installazione*, *Manutenzione*, *Ordine Parti* e *Supporto tecnico* per la classe *Elettrodomestici*.

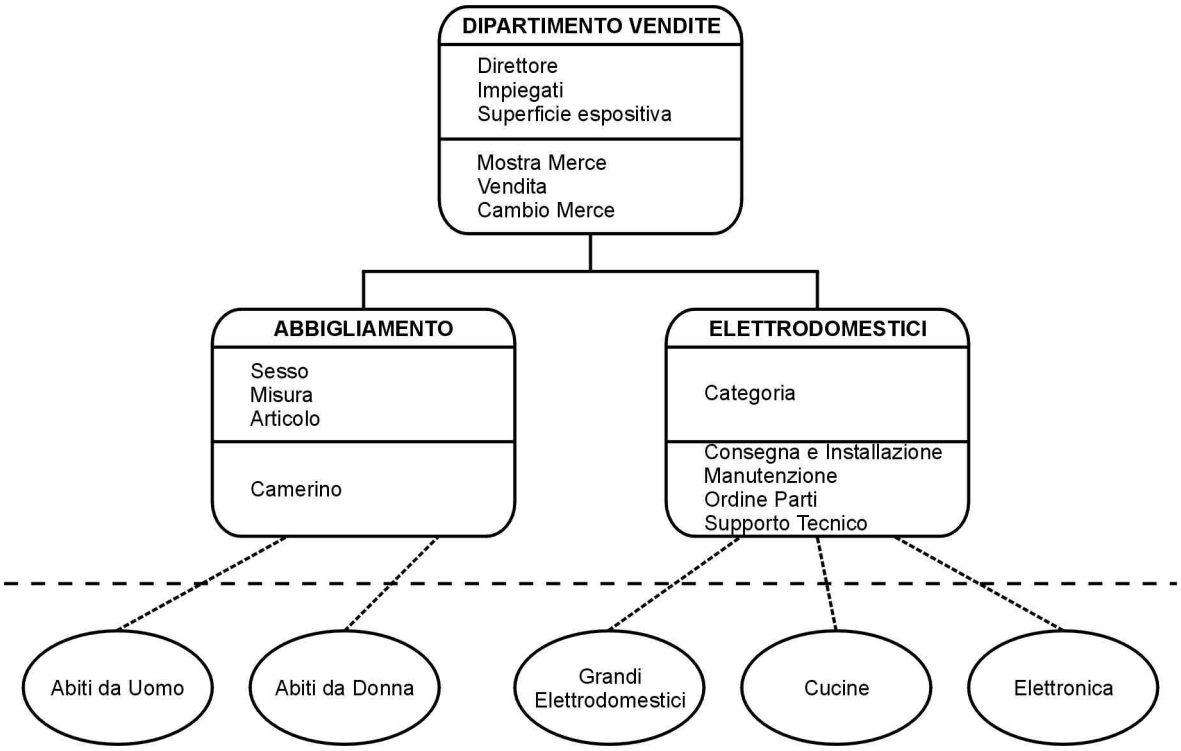


Figura 23: Esempio di applicazione object-oriented

Bibliografia

Parte di questa bibliografia è tratta da: “Object-Oriented Metrics: an Annotated Bibliography”
Empirical Software Engineering Research Group - Department of Computing - Bournemouth University - Poole BH12 5BB, UK

1. Aarsten, A. and M. Morisio. 'Using object oriented technology to measure the software process', in Proc. Proc. 3rd IFIP Int. Conf. on Achieving Quality in Software. Firenze, Italy: 1996.
2. Abbott, D. H., T. D. Korson, et al., A proposed design complexity measure for object-oriented development. No. TR 94-105, Clemson University, 1994.
3. Abernethy, K. and J. C. Kelly. 'Comparing object-oriented and data flow models-a case study', in Proc. 1992 ACM Computer Science Conference. ACM, New York, NY, USA, 1992.
4. Abreu, F. B. and R. Carapuca, 'Candidate Metrics for Object-Oriented Software within a Taxonomy Framework', J. of Systems & Software, 26, pp87-96, 1994.
5. Abreu, F. B. and W. Melo. 'Evaluating the impact of object-oriented design on software quality', in Proc. 3rd International Software Metrics symposium. Berlin, Germany: IEEE Computer Society Press, 1996.
6. Abreu, F. B. 'Metrics for Object-Oriented Development', in Proc. Proc. 3rd International Conference on Software Quality. Lake Tahoe, Nevada, USA: 1993.
7. Abreu, F. B. 'MOOD - Metrics for Object-Oriented Design', in Proc. OOPSLA'94 Workshop Paper Presentation. 1994.
8. Abreu, F. B. 'Object-oriented software engineering: measuring and controlling the development process', in Proc. 4th International Conference on Software Quality. Washington DC: American Society of Quality Control, 1994.
9. Abreu, F. B., 'Design Quality Metrics for Object-Oriented Software Systems', ERCIM News, (23), 1995.
10. Abreu, F. B., M. Goulao, et al. 'Toward design quality evaluation of object-oriented software systems', in Proc. 5th International conference on software quality. Austin, Texas, USA: 1995.
11. Abreu, F. B., R. Esteves, et al. 'The design of Eiffel programs: quantitative evaluation using the MOOD metrics', in Proc. TOOLS'96. Santa Barbara, California, USA: 1996.
12. Abreu, F. B., R. Esteves, et al., The design of Eiffel programs: quantitative evaluation using the MOOD metrics. Technical Report , INESC, 1995.
13. Achee, B. L. and D. L. Carver. 'Evaluating the quality of reverse engineered object-oriented designs', in Proc. 1997 IEEE Aerospace Conference. Proceedings. IEEE, New York, USA, 1997.
14. Ada, Ada and C++: a business case analysis. No. 1-800-AdaIC-11, Ada Information Clearinghouse, 1991.
15. Adams, S. 'OO metrics to investigate', in Proc. OOPSLA'92 Workshop: Metrics for Object-Oriented Software Development. Vancouver, Canada: 1992.
16. Agresti, W. W. and W. M. Evanco, 'Projecting software defects from analysing Ada designs', IEEE Transactions on Software Engineering, 18(11), pp988-997, 1992.
17. Alagic, S. 'Polymorphic and Reflective Type Structures', in Proc. Technology of Object-Oriented Languages and Systems (TOOLS 8). Santa Barbara, USA: Prentice Hall, 1992.
18. Al-Janabi, A. and E. Aspinall, 'An evaluation of software design using the DEMETER tool', Software Engineering Journal, 8(6), pp319-324, 1993.
19. Alkadi, G. and D. L. Carver. 'Application of metrics to object-oriented designs', in Proc. 1998 IEEE Aerospace Conference Proceedings. IEEE, New York, NY, USA, 1998.
20. Alonso, F., J. L. Fuertes, et al. 'A quality model: how to improve the object-oriented software process', in Proc. SMC'98 (1998 IEEE International Conference on Systems, Man, and Cybernetics). IEEE, New York, NY, USA, 1998.
21. Aman, M., T. Yanaru, et al., 'A metric for class structural complexity focusing on relationships among class members', IEICE-Transactions-on-Information-and-Systems., E81-D(12), pp1364-1373, 1998.
22. Ammann, M. H. and R. D. Cameron. 'Measuring program structure with inter-module metrics', in Proc. Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94). IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1994.
23. Anderson, B., Architecture and maintenance in object technology. Unpublished report , Electronic Systems Engineering, University of Essex, UK, 1994.
24. Andrade, R. and G. H. Travassos. 'Principles, guidelines and metrics: their relationship and application to reduce structural complexity of object oriented design', in Proc. European Software Measurement Conference. FESMA 98. Technologisch Instituut Vzw, Antwerpen, Germany, 1998.
25. Antoniol, G., F. Calzolari, et al. 'Adapting function points to object oriented information systems', in Proc. Advanced Information Systems Engineering. 10th International Conference, CAiSE'98. Springer-Verlag, Berlin, Germany, 1998.
26. Antoniol, G., R. Fiutem, et al. 'Using metrics to identify design patterns in object-oriented software', in Proc. Proceedings Fifth International Software Metrics Symposium. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
27. "Archer, C. and M. Stinson, Object-oriented software measures. Tech. report No. SEI-95-TR-002, Software Engineering Institute, Carnegie Melon University, 1995. on availability. "
28. Archer, C., Measuring object-oriented software products. Tech. report No. SEI-CM-028, Software Engineering Institute, Carnegie Melon University, 1995.
29. Armour, F., B. Catherwood, et al. 'Experiences Measuring Object Oriented System Size with Use Cases', in Proc. ESCOM. Wilmslow, UK: 1996.
30. Arnold, M. and P. Pedross. 'Software size measurement and productivity rating in a large-scale software development department', in Proc. Proceedings of the 1998 International Conference on Software Engineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
31. "Arthur H. Watson, Thomas J. McCabe, ""Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric"", NIST Special Publication 500-235"
32. Avotins, J. and C. Mingins. 'Metrics for Object-Oriented Design', in Proc. TOOLs 12. Melbourne: Prentice Hall, 1993.
33. Avotins, J. 'Towards an object-oriented metric modeling method', in Proc. OOPSLA'96 - workshop: OO product metrics. 1996.
34. "AVOTINS, J.. ""Defining and Designing a Quality OO Metrics Suite"". TOOLS 1994. Prentice Hall."

35. Ayerbe, A. and I. Vazquez. 'Software products quality improvement with a programming style guide and a measurement process', in Proc. The Twenty-Second Annual International Computer Software and Applications Conference (Compsac '98). IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
36. Balasubramanian, N. V. 'Object-Oriented Metrics', in Proc. Asia-Pacific Software Engineering Conference. IEEE, 1996.
37. Bandi, R. K. and V. K. Vaishnavi. 'Validating object-oriented design complexity metrics', in Proc. OTC '96. Object Technology Centers. Comsoft, Collegedale, TN, USA, 1996.
38. Banker, R. D., R. J. Kauffman, et al. 'Output measurement metrics in an object-oriented computer aided software engineering (CASE) environment: critique, evaluation and proposal', in Proc. Proceedings of the 24th Hawaii International Conference on System Science,. Hawaii: IEEE Computer Society Press, 1991.
39. Banker, R. D., R. J. Kauffman, et al., 'An empirical test of object-based output measurement metrics in a computer aided software engineering (CASE) environment', Journal of Management Information Systems, 8(3 Winter 1991-92), pp127-150, 1991.
40. Banker, R. D., R. J. Kauffman, et al., 'Automating output size and reuse metrics in a repository-based computer-aided software engineering (CASE) environment', IEEE Transactions on Software Engineering, 20(3), pp169-187, 1994.
41. Banker, R. D., R. J. Kauffman, et al., 'Repository evaluation of software reuse', IEEE Trans. on Software Eng., 19(4), pp379-389, 1993.
42. Banker, R. D., S. M. Datar, et al., 'A model to evaluate variables impacting the productivity of software maintenance projects', Management Science, 37(1), 1991.
43. Bansiya, J. and C. Davis, An object-oriented design quality assessment model. , University of Alabama, 1997.
44. Bansiya, J. and C. Davis, 'Automated metrics and object-oriented development', Dr. Dobbs Journal, (December), pp42-48, 1997.
45. Bansiya, J., L. Etzkorn, et al., 'A class cohesion metric for object oriented designs', Journal of object-oriented programming, 11(8), pp47-52, 1999.
46. Barnard, J., 'A New Reusability Metric for Object-Oriented Software', Software Quality Journal, 7(1), 1998.
47. Barnes, M. G. and B. R. Swim, 'Inheriting software metrics', Journal of Object-Oriented Programming, (November-December), pp27-34, 1993.
48. Basili, V. R., L. C. Briand, et al., 'A validation of object-oriented design metrics as quality indicators', IEEE Transactions on Software Engineering, 22(10), pp751-761, 1996.
49. "BASILI, V.R. AND SELBY, R.W., ""Calculation and Use of an Environment's Characteristic Software Metric Set"" Proc. of the Eighth International Conference on Software Engineering, 1985"
50. "BASILI, V.R. ET AL., ""A Validation of Object-Oriented Design Metrics as Quality Indicators"" IEEE Transactions on Software Engineering, vol. 22 1996"
51. Bellizona, R., M. G. Fugini, et al., 'Reusing specifications in OO applications', IEEE Software, 12(March), 1995.
52. Benford, S., E. Burke, et al., Integrating software quality assurance into the teaching of programming, in Software Quality Assurance and Measurement: a Worldwide Perspective, N. E. Fenton, R. W. Whitty and Y. Lizuke, Editors, International Thomson Computer Press.: London, UK, 1995.
53. Benlarbi, S. and W. Melo, Measuring Polymorphism and Common Interconnections in OO Software Systems. Technical Report No. CRIM-97/11-84, Centre de Recherche Informatique de Montreal, 1997.
54. Berard, E. V., 1996. Metrics for object-oriented software engineering online], Available from: http://www.toa.com/pub/html/moose.html23/07/96].
55. Berard, E. V., Essays on object-oriented software engineering. Prentice Hall: 1993.
56. Berg, W., M. Cline, et al., 'Lessons learned from the OS/400 object oriented project', Commun. ACM, 38(10), pp54-64, 1995.
57. Bertolino, A., R. Mirandola, et al. 'A case study in branch testing automation', in Proc. 3rd International Conference on Achieving Quality in Software. Chapman and Hall, London, 1996.
58. Bezant, SwisRAD Metrics and the Repository. , Bezant Object Technology, 1995.
59. Bieman, J. M. and B. K. Kang. 'Cohesion and reuse in an object oriented system', in Proc. Proc. ACM Software Reusability Symp. (SRS'94),. 1995.
60. Bieman, J. M. and J. X. Zhao. 'Reuse through inheritance: a quantitative study of C++ software', in Proc. Proc. ACM Software Reusability Symp. (SSR'95). 1995.
61. Bieman, J. M. and L. M. Ott, 'Measuring functional cohesion', IEEE Transactions on Software Engineering, 20(8), pp644-657, 1994.
62. Bieman, J. M. and S. Karunanithi, 'Measurement of language supported reuse in object oriented and object based software', J. Systems and Software, 30(3), pp271-293, 1995.
63. Bieman, J. M., Deriving measures of software reuse in object-oriented systems. Technical Report No. CS-91-112, Department of Computer Science, Colorado State University, 1991.
64. Bieman, J. M., Deriving measures of software reuse in object oriented systems, in Formal Aspects of Measurement, T. Denvir, R. Herman and R. Whitty, Editors, Springer-Verlag: London, 1992.
65. Bieman, J. M., Metric development for object-oriented software, in Software Measurement: Understanding Software Engineering, A. Melton, Editor, Chapman and Hall: Kansas City, Missouri., 1995.
66. Bilow, S. C. and B. Henderson-Sellers, 'Report on the Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics , October 23, 1994', Q Methods Report, Committee on Quantitative Methods, Technical Council on Software Engineering, (No. 7, Winter 1995), 1995.
67. Bilow, S. C. and D. Lea. 'Workshop Report: Processes and Metrics for OO Software Development', in Proc. OOPSLA '93 Workshop Report: Processes and Metrics for OO Software Development. 1993.
68. Bilow, S. C. 'Applying graph theoretical analysis to object-oriented system models', in Proc. OOPSLA'92 Workshop: Metrics for Object-Oriented Software Development. Vancouver, Canada: 1992.
69. Bilow, S. C. 'Borrowing from McCabe: what object-oriented methodologists can learn from cyclomatic complexity', in Proc. OOPSLA '92 Workshop 'Metrics for Object-Oriented Software Development'. 1992.
70. Bilow, S., 'Software entropy and the need for object-oriented software metrics', JOOP, 5(8), pp6, 1993.
71. Binder, R. V., 'Design for testability in object-oriented systems', Communications of the ACM, 37(9), pp87-101, 1994.
72. Binkley, A. B. and R. Schach. 'Impediments to the effective use of metrics within the object-oriented paradigm', in Proc. OOPSLA'96 - workshop: OO product metrics. 1996.
73. Binkley, A. B. and S. R. Schach, 'A Comparison of Sixteen Quality Metrics for Object-Oriented Design', Information Processing Letters, 58(6), pp271-275, 1996.
74. Blake, B. A. and P. Jalics, 'An assessment of object-oriented methods and C++', J. Object-oriented Programming, 9(1), pp42-48, 1996.
75. Boehm, B. W., B. K. Clark, et al., An overview of the COCOMO 2.0 software cost model. Technical Report , University of

Southern California, 1995.
76. Booch, G. and M. Vilot, 'Simplifying the Booch components', C++ Report, 5(5), pp41-52, 1993.
77. Booch, G., 'Measures of goodness', Report on Object Analysis and Design (ROAD), 1(2), pp8-10, 14, 1994.
78. Booch, G., Object-Oriented Analysis and Design with Applications. 2nd Edition ed. Benjamin Cummins Publishing Co.: Redwood City, LA, 1994.
79. Booch, G., 'Qualitaetsmasse', OBJEKTspektrum, (September-October), pp53-56, 1994.
80. "BOOCH, GRADY, ""Object Oriented Analysis and Design with Applications"" The Benjamin/Cummings Publishing Company, Inc., 1994. "
81. Bradley, L. 'Evaluating complex properties of object-oriented design and code', in Proc. International Software Quality Conference. Dayton, Ohio: 1991.
82. Briand, L. C., C. Bunse, et al. 'An experimental comparison of the maintainability of object-oriented and structured design documents', in Proc. Intl. Conf. on Software Maintenance. IEEE Computer Society Press, 1997.
83. Briand, L. C., J. Daly, et al. 'A comprehensive empirical validation of design measures for object-oriented systems', in Proc. 5th Intl. Metrics Symp. Bethesda, MD: IEEE Computer Society, 1998.
84. Briand, L. C., J. Daly, et al. 'A unified framework for cohesion measurement in object-oriented systems', in Proc. 4th Intl. Software Metrics Symp. Alberquerque: IEEE Computer Society Press, 1997.
85. Briand, L. C., J. Daly, et al. 'Predicting fault-prone classes with design measures in object-oriented systems', in Proc. Proceedings Ninth International Symposium on Software Reliability Engineering. IEEE Comput. Soc, 1998.
86. Briand, L. C., J. W. Daly, et al., A comprehensive empirical validation of project measures for object-oriented systems. , Fraunhofer Institute forExperimental Software Engineering, Kaiserslautern, 1998.
87. Briand, L. C., J. W. Daly, et al., A unified framework for cohesion measurement in object-oriented systems. No. ISERN-97-05, Fraunhofer Institute (IESE), 1997.
88. Briand, L. C., J. W. Daly, et al., A Unified Framework for Coupling Measurement in Object-Oriented Systems. Technical Report No. ISERN-96-14,, Fraunhofer Institute for Experimental Software Engineering, 1996.
89. Briand, L. C., J. W. Daly, et al., 'A unified framework for cohesion measurement in object-oriented systems', Empirical-Software-Engineering, 3(1), pp65-117, 1998.
90. Briand, L. C., J. W. Daly, et al., 'A unified framework for coupling measurement in object-oriented systems', IEEE-Transactions-on-Software-Engineering, 25(1), pp91-121, 1999.
91. Briand, L. C., K. El Emam, et al., Theoretical and empirical validation of software product measures. No. ISERN-95-03, International Software Engineering Research Network, 1995.
92. Briand, L. C., K. El Emam, et al., 'On the application of measurement theory in software engineering', Empirical Software Engineering: an international journal, 1(1), 1996.
93. Briand, L. C., P. Devanbu, et al. 'An investigation into coupling measures for C++', in Proc. 19th Intl. Conf. on Softw. Eng. Boston, MA: IEEE Computer Society Press, 1997.
94. Briand, L. C., S. Morasca, et al. 'Assessing software maintainability at the end of high-level design', in Proc. IEEE Conference on Software Maintenance. Montreal, Quebec, Canada: 1993.
95. Briand, L. C., S. Morasca, et al., Defining and validating high-level design metrics. Technical Report No. CS-TR 3301, UMIACS-TR-94-75, University of Maryland, 1994.
96. Briand, L. C., S. Morasoa, et al. 'Measuring and Assessing Maintainability at the End of High Level Design', in Proc. Conference on Software Maintenance. Montreal, Canada: IEEE, 1993.
97. Briand, L., C. Bunse, et al., An experimental comparison of the maintainability of object-oriented and structured design documents. No. ISERN-96-13, 1997.
98. Bristol, S. R. 'Tools for object-oriented (OO) metircs collection', in Proc. OOPSLA'96 - workshop: OO product metrics. 1996.
99. Brooks, C. L. and C. G. Buell. 'A tool for automatically gathering object-oriented metrics', in Proc. Aerospace and electronics conference. 1994.
100. Brooks, I. 'Object-oriented metrics collection and evaluation with a software process', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
101. Brownlow, L. S. 'Early Estimating in the Object-Oriented Analysis and Design Environment', in Proc. European Software Cost Modelling Conference. Ivrea, Italy: 1994.
102. Bucci, G., F. Fioravanti, et al. 'Metrics and tool for system assessment', in Proc. Fourth IEEE International Conference on Engineering of Complex Computer Systems. IEEE Comput. Soc, 1998.
103. Burbeck, S. L., 'Real-Time Complexity Metrics for Smalltalk Methods', IBM Systems Journal, 35(2), pp204-226, 1996.
104. Buth, A., 'Softwaremetriken für objekt-orientierte programmiersprachen', Arbeitspapiere der GMD (545), (June), 1991.
105. Byard, C., 'Software beans: class metrics and the mismeasure of software', Journal of Object-Oriented Programming, 7(5), pp32-34, 1994.
106. Caldiera, G., G. Antoniol, et al. 'Definition and experimental evaluation of function points for object-oriented systems', in Proc. 5th Intl. Metrics Symp. Bethesda, MD: IEEE Computer Society, 1998.
107. Campanai, M. and P. Nesi. 'Supporting object-oriented design with metrics', in Proc. Technology of Object-Oriented Languages and Systems, TOOLS 13. Prentice Hall, Hemel Hempstead, UK, 1994.
108. Cant, S. N., B. Henderson-Sellers, et al., 'Application of cogitive complexity metrics to object-oriented programs', Journal of Object Oriented Programming, 8(4), pp52-63, 1994.
109. Cant, S. N., D. R. Jeffery, et al., A conceptual model of cognitive complexity of elements of the programming process. B Centre for Information Technology Research Report No. 57, University of New South Wales, 1992.
110. Cant, S. N., D. R. Jeffery, et al., 'A conceptual model of cognitive complexity of elements of the programming process', Information & Software Technology, 37(7), pp351-362, 1995.
111. Capper, N. P., R. J. Colgate, et al., 'The impact of object-oriented technology on software quality: three case histories', IBM Systems Journal, 33(1), pp131-157, 1994.
112. Cartwright, M. and M. Shepperd, An Empirical Study of Object-Oriented Metrics. Technical Report No. TR 97/01, Dept. of Computing, Bournemouth University, UK, 1997.
113. Cartwright, M. and M. Shepperd, An Empirical View of Inheritance. No. TR98-02, Empirical Software Engineering Group, Bournemouth University, 1998.
114. Cartwright, M. and M. Shepperd, 'An Empirical View of Inheritance', Information & Software Technology, 40(14), pp795-799, 1998.
115. Cartwright, M. and M. Shepperd. 'Building predictive models from object-oriented metrics', in Proc. 8th European Software Control and Metrics Conf. Berlin: 1997.
116. Chae, H. S. and Y. R. Kwon. 'A cohesion measure for classes in object-oriented systems', in Proc. Fifth International Software

Metrics Symposium. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
117. Chan, E., 'A metrics based intelligent tutoring system for object-oriented design', International Journal of Applied Software Technology, 3(4), pp253-272, 1997.
118. Chaudhri, A. B., 'An annotated bibliography of benchmarks for object databases', ACM SIGMOD Record, (March), 1995.
119. Chaumun, M. A., H. Kabaili, et al. 'A change impact model for changeability assessment in object-oriented software systems', in Proc. Third European Conference on Software Maintenance and Reengineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.
120. Chen, D. J. and D. T. K. Chen, 'An experimental study of using reusable software design frameworks to achieve software reuse', Journal of Object-Oriented Programming, 7(2), pp56-67, 1994.
121. Chen, D. J. and P. J. Lee, 'On the study of software reuse using reusable C++ components', Journal of Systems and Software, 20(1), pp19-36, 1993.
122. Chen, D. J. and P. J. Lee, 'On the study of software reuse using reusable C++ components', Journal of Systems and Software, 20(1), pp19-36, 1993.
123. Chen, J. Y. and J. F. Lu, 'A new metric for object-oriented design', Information & Software Technology, 35(4), pp232-240, 1993.
124. Cheol, J. Y., 'A factor analysis of object-oriented software complexity measures for small size C++ programs', Journal of KISS, 23(7), pp711-720, 1996.
125. Chidamber, S. R. and C. F. Kemerer, A metrics suite for object oriented design. Technical Report No. WP No. 249, Center of Information Systems Research (MIT), 1993.
126. Chidamber, S. R. and C. F. Kemerer, 'A metrics suite for object oriented design', IEEE Transactions on Software Engineering, 20(6), pp476-493, 1994.
127. "Chidamber, S. R. and C. F. Kemerer, 'Reply To Comments on """"A Metrics Suite for Object-Oriented Design""""", IEEE Transactions on Software Engineering, 21(4), pp265, 1995."""
128. Chidamber, S. R. and C. F. Kemerer. 'MOOSE: Metrics for Object Oriented Software Engineering', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
129. Chidamber, S. R. and C. F. Kemerer. 'Towards a Metrics Suite for Object Oriented Design', in Proc. OOPSLA '91. ACM, 1991.
130. Chidamber, S. R., D. P. Darcy, et al., Managerial use of object oriented software metrics. Working Paper No. 750, Katz Graduate School of Business, Univ. of Pittsburgh, 1997.
131. Chidamber, S. R., D. P. Darcy, et al., 'Managerial use of object oriented software metrics: an exploratory analysis', IEEE Transactions on Software Engineering, 24(8), pp629-639, 1998.
132. Chung, C. M. and M. C. Lee, 'Inheritance-based metric for complexity analysis in object-oriented design', Journal of Information Science and Engineering, 8(3), pp431-447, 1992.
133. Chung, C. M. and M. C. Lee, 'Object-oriented programming software metrics', International Journal of Mini and Microcomputing, 16(1), pp7-15, 1994.
134. Chung, C. M. and M. C. Lee. 'Inheritance-Based Object-Oriented Software Metrics', in Proc. TENCON '92 IEEE Region 10 Conference. Melbourne, Australia: IEEE, 1992.
135. Chung, C. M., T. K. Shih, et al. 'A graph-theoretical metric for object-oriented software complexity', in Proc. Proceedings of the Eleventh International Symposium on Computer and Information Sciences. ISCIS. Middle East Tech. Univ, Ankara, Turkey: 1996.
136. Chung, C. M., T. K. Shih, et al. 'Inheritance hierarchy metric', in Proc. Eleventh International Symposium on Computer and Information Sciences. ISCIS. Middle East Tech. Univ, Ankara, Turkey: 1996.
137. Chung, C. M., T. K. Shih, et al., 'Object-oriented software development techniques-combining testing and metrics', International Journal on Information and Management Sciences, 6(4), pp17-34, 1995.
138. Chung, C. M., T. K. Shih, et al., 'Object-oriented software testing and metric in Z specification', Information Sciences, 98(1-4), pp175-202, 1997.
139. Chung, C., C. Wang, et al. 'Class Hierarchy Based Metrics for Object-Oriented Design', in Proc. Proceedings of the 1994 IEEE Region 10's Ninth International Conference. Frontiers of Computer Technology. Singapore: IEEE, 1994.
140. Chung, C., T. Shih, et al., 'Integration Object-Oriented Testing and Metrics', International Journal of Software Engineering and Metrics, 7(1), pp125-144, 1997.
141. "Churcher, N. and M. J. Shepperd, 'Comment on """"A metrics suite for object oriented design""""", IEEE Transactions on Software Engineering, 21(3), pp263-265, 1995."""
142. Churcher, N. and M. J. Shepperd. 'Object-Oriented Software Metrics: Holy Grail or Poison Chalice?', in Proc. OOPSLA'94, Metrics Workshop Position Paper. Portland, Oregon: ACM, 1994.
143. Churcher, N. I. and M. J. Shepperd, 'Towards a Conceptual Framework for Object Oriented Software Metrics', ACM SIGSOFT Software Engineering Notes, 20(2), pp69-76, 1995.
144. Cimitile, A., A. de Lucia, et al. 'Identifying objects in legacy systems', in Proc. Proceedings Fifth International Workshop on Program Comprehension. IWPC'97. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1997.
145. Cimitile, A., A. de Lucia, et al., 'Identifying objects in legacy systems using design metrics', Journal of Systems and Software, 44(3), pp199-211, 1999.
146. Ciupke, O. 'Analysis of object-oriented programs using graphs', in Proc. Object-Oriented Technology, ECOOP'97 Workshops Proceedings. Springer-Verlag, Berlin, Germany, 1998.
147. Coad, P. and E. Yourdon, Object-Oriented Design. Yourdon Press/Prentice Hall: 1991.
148. Coffee, P. 'Objects at work: striking the cost/benefit balance', in Proc. 1992 Borland International Developers' Conference. 1992.
149. Condry, C., 'A decade of object-oriented programming books', Journal of Object-Oriented Programming, 7(5), pp59-68, 1994.
150. Constantine, L., 'Efficient Objects', Object Magazine, 7(7), pp71-72, 18, 1997.
151. Consultants, Special issue on object technology. , 1995.
152. Coppick, J. C. and T. J. Cheatham. 'Software metrics for object-oriented systems', in Proc. Annu. ACM Computer Science Conf. Kansas: ACM Press, 1992.
153. Cruz, C. D. 'A proposal of an object oriented development cost model', in Proc. European Software Measurement Conference. FESMA 98. Technologisch Instituut Vzw, Antwerpen, Germany: 1998.
154. CTR, Directory of Object Technology. Vol. ISBN 1-884842-08-9. SIGS Books: 1995.
155. Daly, J., Replication and multi-method approach to empirical software engineering research. Doctoral Thesis , Univ. of Strathclyde, 1996.
156. Daly, J., A. Brooks, et al. 'Evaluating the effect of inheritance on the maintainability of object-oriented software', in Proc. Empirical Studies of Programmers. Washington, DC: 1996.
157. Daly, J., A. Brooks, et al. 'The effect of inheritance on the maintainability of object-oriented software: an empirical study', in

Proc. Intl. Conf. on Softw. Maintenance (ICSM'95). Nice: IEEE Computer Society Press, 1995.
158. David Longstreet, Function Points Step by Step (http://www.softwaremetrics.com/fpsteps.htm)
159. Davis, J. and T. Morgan, 'Object-oriented development at Brooklyn Union Gas', IEEE Software, 10(1), pp67-74, 1993.
160. de Almeida, M. A., H. Lounis, et al. 'An Investigation on the Use of Machine Learned Models for Estimating Software Correctability', in Proc. 20th IEEE Intl. Conf. on Softw. Eng. Kyoto, Japan: IEEE Computer Society Press, 1998.
161. de Champeaux, D., A. Anderson, et al. 'Case study of object-oriented software development', in Proc. OOPSLA. 1991.
162. de Champeaux, D., A. Anderson, et al., Case study of object-oriented software development. No. Report no. HPL-91-170, Hewlett-Packard Lab., 1991.
163. de Champeaux, D., Object-oriented development process and metrics. Prentice Hall International: Englewood Cliffs, NJ, 1997.
164. Debnath, N. C., S. Islam, et al. 'An automated software model generator', in Proc. International Conference on Computers and Their Applications. Cary, NC, USA: Int. Soc. Comput. & Their Appl.-ISCA, 1998.
165. Deubler, H. H. and M. Koestler, 'Introducing object-orientation into large and complex systems', IEEE Trans. on Software Eng., 20(11), pp840-848, 1994.
166. Doake, J. and I. Duncan. 'Amber metrics for the testing and maintenance of object-oriented designs', in Proc. Second Euromicro Conference on Software Maintenance and Reengineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
167. Dorman, M. 'Unit testing of C++ objects', in Proc. EuroSTAR. London: 1993.
168. "Dr. Linda H. Rosenberg, ""Applying and Interpreting Object Oriented Metrics"", Track 7 - Measures/Metrics ATC, NASA"
169. Dumke, R. and A. Winkler. 'The nonclassical use of software metrics', in Proc. Int. Conf. on Software Quality. Research Triangle Park: 1992.
170. Dumke, R. and H. Zuse. 'Software-metriken in der objektorientierten software-entwicklung', in Proc. Deutsche Hochschulschriften 561. HVSnsel, Hohenhausen, Egelsdbach, Frankfurt: 1994.
171. Dumke, R. and I. Kuhrau. 'Tool-based quality management in object-oriented software development', in Proc. QSDT'94. Washington: 1994.
172. Dumke, R. R. 'An object-oriented software measurement and evaluation framework and its application in an industrial environment', in Proc. CONQUEST '98. Conference on Quality Engineering in Software Technology. ASQF, Erlangen, Germany, 1998.
173. Dumke, R. R. and A. S. Winkler. 'Managing the component-based software engineering with metrics', in Proc. Fifth International Symposium on Assessment of Software Tools and Technologies. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1997.
174. Dumke, R. R. and A. S. Winkler. 'Object-oriented software measurement in an OOSE paradigm', in Proc. Software Measurement and Management. Int. Function Point Users Grup, Westerville, OH, USA, 1996.
175. Dumke, R. R. and E. Foltin. 'Metrics-based evaluation of object-oriented software development methods', in Proc. Second Euromicro Conference on Software Maintenance and Reengineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
176. Dumke, R. R., E. Foltin, et al. 'Measurement-based quality assurance in object-oriented software development', in Proc. International Conference on Object Oriented Information Systems, OOIS '95. Springer-Verlag, Berlin, Germany, 1996.
177. Dumke, R., Experiences in object-oriented software development - the empirical basis for object-oriented software metrics. Research Report No. IRB 002/94, University of Magdeburg , Germany, 1994.
178. Dumke, R., Software metrics: a subdivided bibliography. Research Report No. IRB 007/92, University of Magdeburg, Germany, 1993.
179. Dumke, R., E. Foltin, et al. 'Measurement based quality assurance in object-oriented software development', in Proc. OOIS'95. Dublin City University: 1995.
180. Dumke, R., K. Neumann, et al., 'The metric based compiler - a current requirement', ACM SIGPLAN Notices, 27(12), pp29-38, 1992.
181. Durnota, B. and C. Mingins. 'Tree-Based Coherence Metrics in Object-Oriented Design', in Proc. TOOLS 12. 1992.
182. Dvorak, J., 'Conceptual Entropy and Its Effect on Class Hierarchies.', IEEE Computer, 27(June), pp59-63, 1994.
183. "E.V. BERARD, ""Essays on Object-Oriented Software Engineering"", Volume 1, Prentice Hall, Englewood 1993"
184. Ebert, C. and I. Morschel, 'Metrics for quality analysis and improvement of object-oriented software', Information & Software Technology, 39(7), pp497-509, 1997.
185. Eckert, G. and P. Golder, 'Improving object-oriented analysis', Information & Software Technology, 36(2), pp67-86, 1994.
186. Eder, J., G. Kappel, et al., Coupling and cohesion in object-oriented systems. Tech. Rep. , University of Klagenfurt, 1994.
187. Ejiogu, L. O., 'On diminishing the vibrant confusion in software metrics', SIGPLAN Notices, 32(2), pp35-38, 1997.
188. El Emam, K. 'The predictive validity criterion for evaluating binary classifiers', in Proc. Fifth International Software Metrics Symposium. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
189. Eldh, S. 'Helping developers do their job better in an OO environment', in Proc. STAR '96: Fifth International Conference, Software Testing, Analysis and Review. Software Quality Engineering, Jacksonville, FL, USA, 1996.
190. Embley, D. W. and S. N. Woodfield. 'Assessing the quality of abstract data types written in Ada', in Proc. 10th Intl. Conf. on Software Eng. Singapore: Computer Society Press, 1988.
191. Embley, D. W., R. B. Jackson, et al., 'OO systems analysis: is it or isn't it?', IEEE Software, 12(July), pp19-33, 1995.
192. Erni, K. and C. Lewerentz. 'Applying design-metrics to object-oriented frameworks', in Proc. 3rd International Software Metrics Symposium. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
193. Ernst, D. and F. Houdek. 'Applying metrics to cross-technical evaluations', in Proc. European Software Measurement Conference. FESMA 98. Technologisch Instituut Vzw, Antwerpen, Germany: 1998.
194. Etzkorn, L. H. and C. G. Davis, 'Automated object-oriented reusable component identification', Knowledge Based Systems, 9(8), pp517-524, 1996.
195. Etzkorn, L. H., C. G. Davis, et al. 'The program analysis tool for reuse: identifying reusable components', in Proc. FLAIRS-98. Proceedings of the Eleventh International Florida Artificial Intelligence Research Symposium Conference. AAAI Press, Menlo Park, CA, USA, 1998.
196. Etzkorn, L., C. Davis, et al., 'A practical look at the lack of cohesion in methods metric', Joop, 11(5), pp27-34, 1998.
197. Etzkorn, L., J. Bansiya, et al., 'Design and code complexity metrics for OO classes', Joop, 12(1), pp35-40, 1999.
198. Evanco, W. M. 'Poisson models for subprogram defect analyses', in Proc. Achieving Quality in Software. Proceedings of the Third International Conference on Achieving Quality in Software. Chapman & Hall, London, UK, 1996.
199. Fatah, A., An iterative model for estimating development effort of object oriented software. , IBM Australian Programming Centre, 1994.
200. Fayad, M. E. and M. Fulghum, Object-oriented Experiences. SIGS Books: New York, 1996.
201. Fenton, N. E., S. L. Pfleeger, et al., 'Science and substance: a challenge to software engineers', IEEE Software, 11(July), pp86-95, 1994.
202. Fetcke, T., A. Abran, et al. 'Mapping the OO-Jacobson Approach into Function Point Analysis', in Proc. TOOLS USA 97:

Technology of Object-Oriented Languages and Systems. Santa Barbara, California: 1997.
203. Fiedler, S. P., 'Object-oriented unit testing', Hewlett-Packard Journal, (April), 1989.
204. Fioravanti, F., P. Nesi, et al. 'A Tool for Process and Product Assessment and C++ Application', in Proc. 2nd Intl. Conf. on Software Maintenance and Reengineering (CSMR98). Florence: 1998.
205. Fioravanti, F., P. Nesi, et al. 'Assessment of system evolution through characterization', in Proc. Proceedings of the 1998 International Conference on Software Engineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
206. Fitzpatrick, J., 'Applying the ABC Metric to C, C++, and Java', C++ Report, , 1997.
207. Frakes, W. B. and C. J. Fox, 'Sixteen questions about software reuse', Commun. ACM, 38(6), pp75-87, 112, 1995.
208. Frakes, W. B., D. J. Lubinsky, et al., 'Experimental evaluation of a test coverage analyzer for C and C++', J. Systems Software, 16, pp135-139, 1991.
209. Fugate, C. S. and G. Teologlou. 'Estimating the cost of object oriented programming', in Proc. European Software Cost Modelling Conference. Ivrea, Italy: 1994.
210. "G. Myers, ""An Extension to the Cyclomatic Measure of Program Complexity"", SIGPLAN Notices, October, 1977."
211. Gannon, J. D., E. E. Katz, et al., 'Metrics for Ada packages: an initial study', Communications of the A.C.M., 29(7), pp616-623, 1986.
212. Gillibrand, D. and K. Liu, 'Quality Metrics for Object-Oriented Design', Journal of Object-Oriented Programming, 10(8), pp56-59, 1998.
213. Godin, R., H. Mili, et al., 'Design of class hierarchies based on concept (Galois) lattices', Theory and Practice of Object Systems, 4(2), pp117-134, 1998.
214. Gowda, R. G. and L. E. Winslow. 'An approach for deriving object-oriented metrics', in Proc. Natl. Aerospace & Electronics Conference. Dayton, OH: IEEE, 1994.
215. Graham, I. 'Progress With Object-Oriented Metrics', in Proc. Object Expo Europe. London: 1995.
216. Graham, I., 'Making Progress in Metrics', Object Magazine, 6(8), pp68-73, 1996.
217. Graham, I., Migrating to Object Technology. Addison-Wesley: 1995.
218. Grechenig, T. and S. Biffl. 'A flexible generator for code measuring tools providing reusability for metric descriptions', in Proc. TOULOSE '92. Fifth International Conference. Software Engineering and Its Applications. EC2, Nanterre, France, 1992.
219. Halladay, S. and M. Wiebel, Object-oriented Software Engineering. R&D Publications Inc. (distributed by Prentice Hall): 1993.
220. Hamilton, J. 'Metrics: where things went wrong', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
221. Hamza, M., B. Lees, et al. 'The Extension of Software Metrics in Object-Oriented Development', in Proc. Proceedings 3rd International Conference on Software Quality Management. Seville, Spain: 1995.
222. Hanson, E. N., T. M. Harvey, et al., 'Experiences in DBMS implementation using an object-oriented persistent programming languages and a database toolkit', SIGPLAN Notices (OOPSLA'91), 26(11), pp314-328, 1991.
223. Harmon, P. and D. A. Taylor, Objects in Action. Commercial Applications of Object-Oriented Technologies. Addison-Wesley: 1993.
224. Harrison, R. 'An assessment of the impact of Inheritance on the Maintainability of Object-Oriented Systems', in Proc. WESS '97. Bari, Italy: 1997.
225. Harrison, R. and L. G. Samaraweera, 'Using test case metrics to predict code quality and effort', Software Engineering Notes, 21(5), pp78-88, 1996.
226. Harrison, R. and R. Nithi. 'An empirical evaluation of object-oriented design metrics', in Proc. OOPSLA'96 - workshop: OO product metrics. 1996.
227. Harrison, R., Metrics for Object-Oriented Systems. , Dept. of Electronics and Computer Science, University of Southampton, UK, 1995.
228. Harrison, R., L. G. Samaraweera, et al., An evaluation of code metrics for object-oriented programs. Internal Report , Dept. of Electronics and Computer Science, University of Southampton, UK, 1994.
229. Harrison, R., L. G. Samaraweera, et al., Comparing programming paradigms: an evaluation of functional and object-oriented programs. Internal Report , Dept. of Electronics and Computer Science, University of Southampton, UK, 1994.
230. Harrison, R., L. G. Samaraweera, et al., 'An evaluation of Code Metrics for Object-Oriented Programs', Information and Software Technology, 38(7), pp443-450, 1996.
231. Harrison, R., R. Nithi, et al. 'An empirical study of a software maintenance process', in Proc. 5th Software Quality Conference. University of Abertay, Dundee, UK: 1996.
232. Harrison, R., S. Counsell, et al. 'An overview of object-oriented design metrics', in Proc. Eighth IEEE International Workshop on Software Technology and Engineering Practice incorporating Computer Aided Software Engineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.
233. Harrison, R., S. Counsell, et al. 'Coupling metrics for object-oriented design', in Proc. 5th Intl. Metrics Symp. Bethesda, MD: IEEE Computer Society, 1998.
234. Harrison, R., S. J. Counsell, et al. 'Empirical assessment of object-oriented design metrics', in Proc. Empirical Aspects of Software Engineering. Univ. of Keele, UK: 1997.
235. Harrison, R., S. J. Counsell, et al., 'An Evaluation of the MOOD Set of Object-Oriented Software Metrics', IEEE Transactions on Software Engineering, 24(6), pp491-496, 1998.
236. Harrison, R., S. J. Counsell, et al., 'An investigation into the applicability and validity of object-oriented design metrics', Empirical Software Engineering, 3(3), pp255-273, 1998.
237. Hassan, J. A. 'Measuring object oriented systems', in Proc. BCS Specialist Group on Object-Oriented Programming Meeting on Managing Object Technology Projects. IBM South Bank, London: 1993.
238. Hatton, L., Automated incremental improvement of software product quality: a case history, in Software Quality Assurance and Measurement: a Worldwide Perspective, N. E. Fenton, R. W. Whitty and Y. Iizuke, Editors, International Thomson Computer Press: London. UK, 1995.
239. Hatton, L., 'Does OO sync with how we think?', IEEE Software, 15(3), pp46-54, 1998.
240. Hatton, L., Safer C: Developing Software for High-integrity and Safety-critical Systems. McGraw-Hill: London, 1994.
241. Haynes, P. and B. Henderson-Sellers, 'Bringing OO projects under control: an output-, cash-, and time-driven approach', American Programmer, 10(11), pp23-31, 1997.
242. Haythorn, W., 'What is Object-Oriented Design', Journal of Object Oriented Programming, (March/April), pp67-78, 1994.
243. Heitz, M. and B. Labreuille. 'Design and development of distributed software using Hierarchical Object Oriented Design and Ada', in Proc. Ada in Industry. Proceedings of the Ada-Europe Int. Conf. Munich: Cambridge University Press, Cambridge, England, 1988.
244. Henderson, R. and B. Zorn, 'A comparison of object-oriented programming in four modern languages', Software Practice and

Experience, 24(11), pp1077-1096, 1994.
245. Henderson-Sellers, B. and J. Edwards, BOOKTWO of Object-Oriented Knowledge: the Working Object. Prentice Hall: Sydney, Australia, 1994.
246. Henderson-Sellers, B. 'Building a metrics programme', in Proc. Object Expo Europe/Java Expo. SIGS Conferences, Newdigate, UK, 1997.
247. Henderson-Sellers, B. 'Identifying internal and external characteristics of classes likely to be useful as structural complexity metrics', in Proc. International Conference on Object Oriented Information Systems (OOIS'94). London: Springer-Verlag, London, 1994.
248. Henderson-Sellers, B. 'Managing an object-oriented metrics programme', in Proc. Object Expo Europe - Java Expo. SIGS Conferences, Newdigate, UK, 1996.
249. Henderson-Sellers, B. 'Managing and measuring object-oriented information systems development', in Proc. Workshop on Object-Oriented Software Engineering Practice. Denver, CO: 1992.
250. Henderson-Sellers, B. 'Some metrics for object-oriented software engineering', in Proc. TOOLS 6. Prentice-Hall, 1991.
251. "HENDERSON-SELLERS, B., ""Object-Oriented Metrics: Measures of Complexity"" Upper Saddle River, NJ:Prentice-Hall, 1996."
252. Henderson-Sellers, B., D. Tegarden, et al. 'Tutorial Notes: Metrics for Object-Oriented Software', in Proc. OOPSLA 94. 1994.
253. Henderson-Sellers, B., D. Tegarden, et al. 'Tutorial: Object-Oriented Metrics', in Proc. TOOLS-Pacific. Sidney, Australia: 1992.
254. Henderson-Sellers, B., 'Generalization of object-oriented components for reuse', Journal of Object Oriented Programming, 9(2), pp19-31, 1997.
255. Henderson-Sellers, B., L. Constantine, et al., 'Coupling and Cohesion (Towards a Valid Metrics Suite for Object-Oriented Analysis and Design)', Object-Oriented Systems, 3(3), pp143-158, 1996.
256. Henderson-Sellers, B., Object-Oriented Metrics: measures of complexity. Prentice Hall: Hemel Hempstead, UK, 1996.
257. Henderson-Sellers, B., 'OO metrics programme', Object Magazine, (October), pp73-79,95, 1995.
258. Henderson-Sellers, B., S. Moser, et al. 'A proposed multi-dimensional framework for object-oriented metrics', in Proc. 1st Australian Software Metrics Conf. 1993.
259. Henderson-Sellers, B., 'The economics of reusing library classes', Journal of Object-Oriented Programming, 6(4), pp43-50, 1993.
260. Henderson-Sellers, B., 'Workshop report : pragmatic and theoretical directions on object-oriented software metrics', OOPS Messenger, 5(4), pp78-80, 1994.
261. Henry, S. and J. Lewis, 'Integrating metrics into a large-scale software development environment', J. of Systems & Software, 13(2), pp89-95, 1990.
262. "HENRY, S. AND LI, W., ""Metrics for Object Oriented Systems"" OOPSLA '92 Workshop: Metrics for Object-Oriented Software Development, 1992"
263. Henry, S. and M. Humphrey. 'A controlled experiment to evaluate maintainability of object-oriented software', in Proc. IEEE Conf. on Softw. Maint. IEEE Computer Society Press, 1990.
264. Henry, S. M. and M. Humphrey, Comparison of an object-oriented programming language to a procedural language for effectiveness in program maintenance. Tech. Report No. TR 88-49, Virginia Polytechnic Inst., Dept of Computer Science, 1988.
265. Henry, S. M. and M. Humphrey, 'Object-oriented vs. procedural programming languages: effectiveness in program maintenance', Journal of Object-Oriented Programming, 6(3), pp41-49, 1993.
266. Henry, S. M. and M. Lettanzi. 'Measurement of software maintenance and reliability in the object oriented paradigm', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
267. Henry, S. M. and W. Li. 'Maintenance Metrics for the Object Oriented
268. Henry, S. M. and W. Li. 'Metrics for object-oriented systems', in Proc. OOPSLA'92 Workshop: Metrics for Object-Oriented Software Development. Vancouver, Canada: 1992.
269. Hericko, M., I. Rozman, et al. 'Integration of strategic, tactical and technical OO measurement', in Proc. Software Quality Management VI. Quality Improvement Issues. Springer-Verlag London, London, UK, 1998.
270. Hericko, M., I. Rozman, et al. 'OO metrics data gathering environment', in Proc. Technology of Object-Oriented Languages. TOOLS 24. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
271. Hericko, M., I. Rozman, et al. 'Tools for OO metrics collection', in Proc. International Conference on Software Quality, ICSQ '97. Drustvo Ekonomistov, Maribor, Slovenia: 1997.
272. Hiekotter, V., B. Helling, et al., 'Design metrics: an aid to their automatic collection', Information & Software Technology, 32(1), pp79-87, 1990.
273. Hitz, M. and B. Montazeri, 'Chidamber & Kemerer's metrics suite: a measurement theory perspective', IEEE Trans. on Softw. Eng, 22(4), pp267-271, 1996.
274. Hitz, M. and B. Montazeri, 'Measuring Coupling in Object-Oriented Systems', Object Currents, 1(4), 1996.
275. Hitz, M. and B. Montazeri. 'Measuring coupling and cohesion in object-oriented systems', in Proc. Intl. Symposium on Applied Corporate Computing. Monterrey, Mexico: 1995.
276. Hitz, M. and B. Montazeri. 'Measuring product attributes of object-oriented systems', in Proc. 5th European Software Engineering Conference (ESEC `95). Barcelona, Spain: Springer Verlag 1995, 1995.
277. Hitz, M. 'Measuring reuse attributes in object-oriented systems', in Proc. OOIS'95. Dublin City University: Springer-Verlag, 1995.
278. Hoffman, D. and P. Strooper, 'The testgraph methodology: automated testing of collection classes', Journal of Object-Oriented Programming, (Nov.-Dec.), 1995.
279. Hopkins, T. P., 'Do we need object-oriented design metrics?', Hotline on Object-Oriented Technology, 2(8), pp16-17, 1991.
280. Hopkins, T., Complexity metrics for quality assessment of object-oriented design, in Software Quality Management II, vol. 2: Building Quality into Software, P. M. Ross, C. A. Brebbia, G. Staples and J. Stapleton, Editors, Computational Mechanics Press: 1994.
281. "Horvilleur, C. and G. Horvilleur. 'Documenting reusable components: an engineering approach', in Proc. Technology of Object-Oriented Languages and Systems TOOLS 5. Proceedings of the Fifth International Conference. Prentice Hall, Engelwood Cliffs, NJ, USA; 1991; viii+485 pp. 1991."""
282. Hsia, P., A. Gupta, et al. 'A study of the effect of architecture on maintainability of object-oriented systems', in Proc. Intl. Conf. on Softw. Maintenance (ICSM'95). Nice, France: IEEE, 1995.
283. Hudli, R. V., C. L. Hoskins, et al. 'Software Metrics for Object-Oriented Design', in Proc. IEEE International Conference on Computer Design, VLSI in Computers and Processors. Los Alamitos, CA, USA: IEEE, 1994.
284. "HUDLI, R., C. HOSKINS, AND A. HUDLI, ""Software Metrics for Object-Oriented Designs"" IEEE, April 1994."
285. Huemer, C., G. Kappel, et al., 'Migration in object-oriented database systems - a practical approach', Software Practice and Experience, 25(10), pp1065-1096, 1995.

286. Huitt, R. 'Metrics for object-oriented software development, Workshop Position Paper', in Proc. OOPSLA'92 Workshop: Metrics for Object-Oriented Software Development. Vancouver, Canada: 1992.
287. Humphrey, W. S., 'Estimating with Objects Parts I-XI', Object Currents/Object Magazine Online, , 1996.
288. Humphrey, W., A Discipline for Software Engineering. Vol. ISBN 0-201-54610-8. Addison-Wesley: Reading, MA, 1995.
289. IEE, IEE Colloquium on 'Object-oriented development'. No. Digest no. 007, IEE, 1993.
290. IEEE Computer, Vol. 25, No. 10, October 1992
291. Iivari, J., 'Object-orientation as structural, functional and behavioural modelling: a comparison of six methods for object-oriented analysis', Information and Software Technology, 37(3), pp155-164, 1995.
292. Iivari, J., Object-oriented information systems analysis: A comparison of six object-oriented analysis methods, in Methods and Associated Tools for the Information Systems Life Cycle, A. A. a. O. Verrijn-Stuart, T.W., Editor, North-Holland: 1994.
293. Jacobson, I., M. Christerson, et al., Object-Oriented Software Engineering: A Use Case Driven Approach. ACM Press/Addison-Wesley: Reading, MA, 1992.
294. Jensen, R. L. and J. W. Bartley, 'Parametric estimation of programming effort: an object-oriented model', J. Systems and Software, 15(2), pp107-114, 1991.
295. Johnson, M. A., 'A case study of tracking software development using quality metrics', Software Quality Journal, 4(1), pp15-31, 1995.
296. Jones, C., Strengths and weaknesses of software metrics. Technical Report , Software Productivity Research, Inc., 1997.
297. Jones, C., The economics of object-oriented software. Technical Report , Software Productivity Research, Inc., 1997.
298. Jones, C., 'Backfiring: converting lines of code to function points', IEEE Computer, 28(11), pp87-88, 1995.
299. Jones, J. 'How Logica Projects Measure Up', in Proc. European COCOMO Users Group Meeting. City University, London: 1992.
300. Judge, T. R. and N. S. Mistry. 'Metrics for estimation', in Proc. Software Quality Management VI. Quality Improvement Issues. Springer-Verlag London, London, UK, 1998.
301. Kaczmarek, J. and M. Kucharski. 'Application of object-oriented metrics for Java programs', in Proc. 10th European Software Control & Metrics Conference. Herstmonceux, England: Shaker Publishing, 1999.
302. Kain, J. B., 'Measuring the ROI of reuse', Object Magazine, 4(3), pp48-54, 1994.
303. Kalakota, R. 'The role of complexity in object-oriented systems development', in Proc. 26th Hawaii International Conference on System Sciences. Hawaii: IEEE Computer Society Press, 1993.
304. Kamiya, T., S. Kusumoto, et al., 'Empirical evaluation of reuse sensitiveness of complexity metrics', Information and Software Technology, 41(5), pp297-305, 1999.
305. Kang, B. K. and J. M. Bieman. 'Inheritance tree shapes and reuse', in Proc. Proceedings. Fourth International Software Metrics Symposium. IEEE Comput. Soc, 1997.
306. Kang, K. C. and L. S. Levy, 'Software methodology in the harsh light of economics', Information and Software Technology, 31(5), pp239-250, 1989.
307. Karlsson, E. A., ed. Software Reuse: a Holistic Approach. J. Wiley: Chichester, 1995.
308. Karunanithi, S. and J. M. Bieman, Measuring Software Reuse in Object Oriented Systems and Ada Software. No. TR CS-93-125, Colorado State University, 1993.
309. Karunanithi, S. and J. M. Bieman. 'Candidate reuse metrics for object oriented and Ada software', in Proc. Proc IEEE-CS Int. Software Metrics Symp. Baltimore, MD: IEEE, 1993.
310. Kauffman, R. and R. Kumar, Modeling estimation expertise in object based CASE environments. , Stern School of Business , New York University, 1993.
311. Kazman, R., L. Bass, et al. 'SAAM: A method for analysing the properties of software architectures', in Proc. 16th Intl. Conf. on Softw. Eng. Sorrento, Italy: IEEE Computer Press, 1994.
312. Kazman, R., L. Bass, et al., An architectural analysis case study: internet information systems. Internal Paper No. 15213. 2, Carnegie Mellon University, 1995.
313. Keyes, J., 'Code trapped between legacy, object worlds', Software Magazine, 12(8), 1992.
314. Keyes, J., 'New metrics needed for new generation', Software Magazine, 12(6), pp42-56, 1992.
315. Kim, E. M., O. B. Chang, et al. 'Analysis of metrics for object-oriented program complexity', in Proc. Eighteenth Annual International Computer Software and Applications Conference (COMPSAC 94). IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1994.
316. Kim, E. M., S. H. Jeon, et al., 'A program complexity metric and its evaluation tool for C++ programs', Journal of KISS, 3(6), pp656-665, 1997.
317. Kim, E. M., S. Kusumoto, et al. 'Heuristics for computing attribute values of C++ program complexity metrics', in Proc. The Twentieth Annual International Computer Software and Applications Conference (COMPSAC '96). IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
318. Kim, K., Y. Shin, et al. 'Complexity measures for object-oriented program based on the entropy', in Proc. 1995 Asia Pacific Software Engineering Conference. IEEE Comput. Soc. Press, Los Alamitos, CA, 1995.
319. Kiran, G. A., S. Haripriya, et al. 'Effect of object orientation on maintainability of software', in Proc. International Conference on Software Maintenance. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.
320. Kirsopp, C., M. Shepperd, et al., An empirical study into the use of measurement to support OO design evaluation. No. TR99-01, Empirical Software Engineering Group, Bournemouth University, 1999.
321. Kokol, P., V. Zumer, et al., 'PROMIS: A Software Metrics Tool Generator', ACM SIGPLAN Notices, 30(5), pp37-42, 1995.
322. Kolewe, R., 'Metrics in object-oriented design and programming', Software Development, (October), pp53-62, 1993.
323. Korson, T. and J. D. McGregor, 'Technical criteria for the specification and evaluation of object-oriented libraries', Software Engineering Journal, 7(3), pp85-94, 1992.
324. KPMG, Object Technology Survey Results. , KPMG Management Consulting, 1994.
325. Kraemer, T. F., 'Product Development Using Object-oriented Software', Hewlett Packard Journal, (August), pp87-100, 1989.
326. Kristen, G., Object Orientation. The KISS Method: From Information Architecture to Information System. Addison-Wesley: Wokingham, UK, 1994.
327. Kung, D., J. Gao, et al., 'Class firewall, test order and regression testing of object-oriented programs', J. Object-Oriented Programming, 8(2), pp51-65, 1995.
328. Kung, D., J. Gao, et al., 'Developing an object-oriented software testing and maintenance environment', Commun. ACM, 38(10), pp75-87, 1995.
329. "L. LARANJEIRA, ""Software Size Estimation of Object-Oriented Systems"" IEEE Transactions on Software Engineering, Vol. 16, No. 5, May 1990"
330. Lahire, P. and J. M. Jugant. 'Lessons learned with Eiffel 3: the K2 project', in Proc. TOOLS 17. Prentice Hall, New York, 1995.
331. Lake, A. and C. Cook. 'A software complexity measure for C++', in Proc. 4th Annual Oregon Workshop on Software Metrics.

Silver Falls, Oregon: 1992.
332. Lake, A. 'Use of factor analysis to develop OOP software complexity metrics', in Proc. 6th Annual Oregon Workshop on Software Metrics. Silver Falls, Oregon: 1994.
333. LaLonde, W. and J. Pugh, 'Gathering metric information using metalevel facilities', Journal of Object-Oriented Programming, (March/April), pp33-37, 1994.
334. Lamping, J. 'Typing the Specialization Interface', in Proc. OOPSLA 93. Addison-Wesley, 1993.
335. Laranjeira, L. A., 'Software size estimation of object-oriented systems', IEEE Trans. Software Eng, 16(5), pp510-522, 1990.
336. Laranjeira, L., 'Software size estimation of object-oriented systems', Software Development, 1(2), pp53-62, 1993.
337. Lee, R. Y., J. P. Kelsh, et al. 'Special metrics for object-oriented software design', in Proc. IASTED/ISMM International Conference Modelling and Simulation. IASTED-ACTA Press, Anaheim, CA, USA, 1996.
338. Lee, Y. S., B. S. Liang, et al. 'Measuring the coupling and cohesion of an object-oriented program based on information flow', in Proc. International Conference on Software Quality 1995, (ICSQ'95). Maribor, Slovenia: 1995.
339. Lee, Y. S., B. S. Liang, et al. 'Some complexity metrics for object-oriented programs based on information flow', in Proc. Computer System and Software Engineering 1993, 7th Annual European Computer Conference, (CompEuro'93). Paris, France: 1993.
340. Lee, Y. S., B. S. Liang, et al., 'Some complexity metrics for object-oriented programs based on information flow: a study of C++ programs', Journal of Information and Software Engineering, 1994(10), pp21-50, 1994.
341. "LEE, Y., B. LIANG, AND F. WANG, ""Some Complexity Metrics for Object-Oriented Programs Based on information Flow"""
342. Lewerentz, C. and F. Simon. 'A product metrics tool integrated into a software development environment', in Proc. European Software Measurement Conference. FESMA 98. Technologisch Instituut Vzw, Antwerpen, Germany: 1998.
343. Lewis, J. A. 'Quantified Object-Oriented Development: Conflict and Resolution', in Proc. 4th Software Quality Conference. Dundee, UK: University of Abertay, Dundee, UK, 1995.
344. Lewis, J. A., S. Henry, et al., 'An empirical study of object-oriented paradigm and software reuse', ACM SIGPLAN Notices, 26(11), pp184-196, 1991.
345. Li, P., J. Kennedy, et al. 'Assessing inheritance for the multiple descendant redefinition problem in OO systems', in Proc. OOIS'97. 1997 International Conference on Object Oriented Information Systems. Springer-Verlag London, London, UK, 1998.
346. Li, P., J. Kennedy, et al. 'Mechanisms for interpretation of OO systems design metrics', in Proc. Technology of Object-Oriented Languages. TOOLS 24. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
347. Li, W. and H. Delugach. 'Software metrics and application domain complexity', in Proc. Asia Pacific Software Engineering Conference and International Computer Science Conference. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.
348. "LI, W. AND HENRY, S., ""Object-Oriented Metrics that Predict Maintainability"" Journal of Systems and Software, vol. 23."
349. Li, W. and S. M. Henry, 'An empirical study of maintenance activities in two object-oriented systems', Journal of Software Maintenance: Research & Practice, 2(2), pp131-147, 1995.
350. Li, W. and S. M. Henry, 'Object-oriented metrics that predict maintainability', J. Systems and Software, 23(2), pp111-122, 1993.
351. Li, W. and S. M. Henry. 'Maintenance metrics for the object oriented paradigm', in Proc. 1st Intl. Software Metrics Symposium. Baltimore, MD: IEEE Computer Society, 1993.
352. Li, W., Applying Software Maintenance Metrics in the Object-Oriented Software Development Life Cycle. Ph.D. Thesis , Virginia Polytechnic Inst. and State Univ., 1992.
353. Li, W., 'Another metric suite for object-oriented programming', Journal of Systems and Software, 44(2), pp155-162, 1998.
354. Li, W., S. M. Henry, et al., 'Measuring object-oriented design', J. Object-oriented Programming, (July/August), 1995.
355. Liddiard, J., 'Achieving testability when using Ada packaging and data hiding methods', Ada User, 14(1), pp27-32, 1993.
356. Lim, W. C., 'Effects of reuse on quality, productivity and economics', IEEE Software, 11(September), 1994.
357. Lindvall, M. and M. Runesson. 'The visibility of maintenance in object models: an empirical study', in Proc. International Conference on Software Maintenance. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
358. Linthicum, D. S., 'In Search of the Elusive OO Metric', Object Magazine, 6(6), pp64-66, 1996.
359. Liu, C. P., B. S. Liang, et al. 'A validation of software complexity metrics for object-oriented programs', in Proc. International Computer Symposium 1994, (ICS'94). Taiwan, R.O.C.: 1994.
360. Liu, C., S. Goetze, et al., 'What contributes to successful object-oriented learning?', ACM SIGPLAN Notices, 27(10), pp77-86, 1992.
361. Lo, B. W. N. and S. Haifeng. 'A preliminary testability model for object-oriented software', in Proc. 1998 International Conference Software Engineering: Education and Practice. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
362. Lorenz, M. and J. Kidd, Object-Oriented Software Metrics. Object-Oriented Series, Prentice Hall: Englewood Cliffs, N.J., 1994.
363. Lorenz, M. and J. Kidd. 'O-O metrics position paper', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
364. Lorenz, M., Object-Oriented Software Development: a Practical Guide. Vol. ISBN 0-13-72692-8. Prentice Hall: Englewood Cliffs, N.J., 1993.
365. Lounis, H. and W. L. Melo. 'Identifying Coupling in Modular Systems', in Proc. 7th Intl. Conf. on Software Technology. Curitiba, Brazil: 1997.
366. Lounis, H., H. A. Sahraoui, et al., Identifying and Measuring Coupling in OO systems. Technical Report No. CRIM-97/11-82, CRIM, 1997.
367. Love, T., Object Lessons: Lessons Learned in Object-Oriented Development Projects. Vol. ISBN 0-9627477-3-4. SIGS Books Inc.: New York, 1993.
368. Lytz, R., 'Software metrics for the Boeing 777: a case study', Software Quality Journal, 4(1), pp1-13, 1995.
369. "M. LORENZ AND J. KIDD, ""Object-Oriented Software Metrics"" Prentice Hall, Englewood Cliffs, New Jersey, 1994."
370. Major, M. L. and J. D. McGregor. 'The application of function point analysis (FPA) to object-oriented software development', in Proc. OTC '96. Object Technology Centers. Comsoft, Collegedale, TN, USA, 1996.
371. Mancl, D. and W. Havanas. 'A study of the impact of C++ on software maintenance', in Proc. IEEE Conf. on Software Maintenance. IEEE Computer Society Press, 1990.
372. Marchesi, M. 'OOA metrics for the Unified Modeling Language', in Proc. Second Euromicro Conference on Software Maintenance and Reengineering. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
373. Martin, R. C., 'Large-scale stability', C++ Report, 9(2), pp54-60, 1997.
374. Martin, R. 'OO design quality metrics - an analysis of dependencies (Position Paper)', in Proc. Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94. 1994.
375. Martin, R., 1994. Object-oriented design quality metrics : an analysis of dependencies online], Available from: http://www.oma.com/Offerings/catalog.html24 June 1996].
376. Martin, R., Designing object oriented C++ applications using the Booch method. Vol. ISBN 0-13-203837-4. Prentice Hall:

Englewood Cliffs, New Jersey, 1995.
377. Maus, A. 'Entropy as a complexity and optimal module size in an object-oriented program', in Proc. Algorithms, Software, Architecture, Information Processing. IFIP 12th World Congress. Elsevier Science Publishers, 1992.
378. Mayrand, J., F. Guay, et al. 'Inheritance graph assessment using metrics', in Proc. 3rd International Software Metrics Symposium. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
379. McCabe, T. J. and A. H. Watson, 'Combining comprehension and testing in object-oriented development', Object Magazine, 4(1), pp63-64, 1994.
380. McCabe, T. J., L. A. Dreyer, et al., 'Testing an object-oriented application', Journal of the Quality Assurance Institute, (October), pp21-27, 1994.
381. McClure, S., Object tools - Smalltalk market accelerates. , International Data Corporation, 1995.
382. McGregor, J. D. and S. Srinivas. 'A measure of testing effort', in Proc. Second USENIX Conference on Object-Oriented Technologies and Systems (COOTS). USENIX Assoc, Berkeley, CA, USA, 1996.
383. McGregor, J., 'Managing metrics in an iterative environment', Object Magazine, (October), pp65-71, 1995.
384. Melo, W. L., L. C. Briand, et al., Measuring the impact of reuse on quality and productivity in object-oriented systems. Technical Report No. CS-TR-3395, University of Maryland, Dept of Computer Science, 1995.
385. Metsker, S. J., 'Object weights and measures', Object Magazine, 6(2), pp83-95, 1996.
386. Meyer, B., 'The role of object-oriented metrics', IEEE Computer, 31(11), pp123-125, 1998.
387. Milankovic-Atkinson, M. and E. Georgiadou. 'Metrics for reuse of object-oriented software', in Proc. Software Quality Management IV. Improving Quality. Mech. Eng. Publications, London, UK, 1996.
388. Milankovic-Atkinson, M. and E. Georgiadou. 'Object-oriented software development with reuse', in Proc. Sixth International Conference on Software Quality Management. Springer-Verlag London, London, UK, 1998.
389. Mili, H., F. Mili, et al., 'Reusing software: issues and research directions', IEEE Trans. on Softw. Eng., 21(6), pp528-562, 1995.
390. Miller, B. K., H. Pei, et al. 'Object-oriented architecture measures', in Proc. 32nd Annual Hawaii International Conference on Systems Sciences. IEEE Comput. Soc, Los Alamitos, CA, USA, 1999.
391. Miller, G., M. Woof, et al., Project MOOD: Business structure. Technical Report No. PS-MOOD-CS-000003, Parallax Solutions Limited, 1996.
392. Miller, G., M. Woof, et al., Project MOOD: Requirements capturing approach. Technical Report No. PS-MOOD-CS-000002, Parallax Solutions Limited, 1996.
393. Miller, G., M. Woof, et al., Project MOOD: Survey results. No. PS-MOOD-CS-000005, Parallax Solutions Limited, 1996.
394. Mingins, C. 'Designing software metrics (Tutorial presentation abstract)', in Proc. TOOLS 17. Prentice Hall , New York, 1995.
395. Mingins, C., B. Durnota, et al. 'Collecting Software Metrics Data for the Eiffel Class Hierarchy', in Proc. TOOLS. 1993.
396. Misic, V. B. and D. N. Tesic, 'Estimation of effort and complexity: an object-oriented case study', Journal of Systems and Software, 41(2), pp133-143, 1998.
397. Misic, V. B. and S. Moser. 'From formal metamodels to metrics: an object-oriented approach', in Proc. Technology of Object-Oriented Languages. TOOLS 24. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
398. Misic, V. B., D. Tesic, et al. 'Downsizing the estimation of software quality: a small object-oriented case study', in Proc. Technology of Object-Oriented Languages. TOOLS 27. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
399. Mitchell, J., J. E. Urban, et al., 'The effect of abstract data types on program development', IEEE Computer, 20(August), pp85-88, 1987.
400. Monarchi, D. E. and G. I. Puhr, 'A Research Typology for Object-Oriented Analysis and Design', Communications of the ACM, 35(9), pp35-47, 1992.
401. Moreau, D. R. and W. D. Dominick, 'A programming environment evaluation methodology for object-oriented systems: part II - test case application', Journal of Object-Oriented Programming, 3(3), pp23-32, 1990.
402. Moreau, D. R. and W. D. Dominick, 'A programming environment evaluation methodology for object-oriented systems: part I - the methodology', Journal of Object-Oriented Programming, 3(3), pp38-52, 1990.
403. Moreau, D. R. and W. D. Dominick, 'Object-oriented graphical information systems: research plan and evaluation metrics', Journal of Systems and Software, 10, pp23-28, 1989.
404. Morisio, M. 'A methodology to measure the software process', in Proc. Annual Oregon Workshop on Software Metrics. Silver Falls, Oregon: 1995.
405. Morisio, M. 'Issues in defining and applying a measurement plan to an object oriented process', in Proc. 4th Software Quality Conference. Dundee, Scotland: University of Abertay, Dundee, UK, 1995.
406. Morisio, M. 'Measuring reuse in an object-oriented process', in Proc. 8th Intl. Conf. on Software Engineering and its Applications. Paris-La-Defense: 1995.
407. Morris, K. L., Metrics for Object-Oriented Software Development Environments. Masters Thesis , M.I.T., 1988.
408. "Morschel, I. and C. Ebert. 'Applying metrics for quality analysis and improvement of object-oriented software', in Proc. 3rd International Conference on Achieving Quality in Software. Chapman and Hall, 1996. "
409. Morschel, I., Applying object-oriented metrics to enhance software quality, in Theorie und Praxis der Softwaremessung, R. Dumke and H. Zuse, Editors, Deutscher Universitaetsverlag: Wiesbaden, Germany, 1994.
410. Moser, S. and O. Nierstrasz, 'The Effects of Object-Oriented Frameworks on Developer Productivity', IEEE Computer, 29(9), pp45-51, 1996.
411. "Moser, S. and V. B. Misic. 'Measuring class coupling and cohesion: a formal metamodel approach', in Proc. Asia Pacific Software Engineering Conference and International Computer Science Conference ;. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997."""
412. Moser, S., B. Henderson-Sellers, et al. 'Measuring object-oriented business models', in Proc. Technology of Object-Oriented Languages and Systems, TOOLS 25. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
413. Moser, S., 'Metamodels for object-oriented systems: a proposition of metamodels describing object-oriented systems at consecutive levels of abstraction', Software Concepts and Tools, 16(2), pp63-80, 1995.
414. Moses, J. 'Re-usability and extendability in object-oriented and object-based design', in Proc. Software Quality Management II. Building Quality into Software. Comp. Mech. Publications, Southampton, UK, 1994.
415. "N.E. FENTON, ""Software Metrics: A Rigorous Approach"", Chapman and Hall 1991."
416. Nakanishi, K. and T. Arano. 'A metric for evaluating effectiveness of object-oriented interface abstraction for promoting software reuse', in Proc. Object Technologies for Advanced Software. Second JSSST International Symposium ISOTAS'96. Springer-Verlag, Berlin, Germany, 1996.
417. Nakanishi, K., T. Arano, et al. 'A metric for evaluating class library interfaces and its application to library upgrades', in Proc. Intl. Conf. on Softw. Maintenance (ICSM'95). Nice, France: IEEE, 1995.
418. Nakanishi, K., T. Arano, et al. 'Understandability of class libraries and Zipf's law', in Proc. Technology of Object-Oriented

Languages and Systems Tools 12. Prentice Hall, Englewood Cliffs, NJ, USA, 1993.
419. Neal, R. D. 'A taxonomy of object-oriented measures', in Proc. Software Quality Engineering. Comput. Mech. Publications, Southampton, UK, 1997.
420. Neal, R. D. 'Modeling the object-oriented space through validated measures', in Proc. 1997 IEEE Aerospace Conference. Proceedings. IEEE, New York, NY, USA, 1997.
421. Neal, R. D. 'The applicability of proposed object-oriented metrics to developer feedback in time to impact development', in Proc. Advances in Concurrent Engineering, CE 96. Third ISPE International Conference on Concurrent Engineering Research: Research and Applications. Technomic Publishing, Lancaster, PA, USA, 1996.
422. Nesi, P. and M. Campanai, 'Metric Framework for Object-Oriented Real-Time Systems Specification Languages', Journal of Systems and Software, 34, pp43-65, 1996.
423. Nesi, P. and T. Querci, 'Effort Estimation and Prediction of Object-Oriented Systems', Journal of Systems and Software, 42(1), pp89-102, 1998.
424. Nesi, P., 'Managing OO projects better', IEEE Software, 15(4), pp50-60, 1998.
425. ObjectSoftware_Inc, 'ObjectDetail - a tool to measure object metrics and error densities in an application', (May), 1995.
426. Offut, A. J. and A. Irvine. 'Testing object-oriented software using the category-partition method', in Proc. TOOLS 17. Prentice Hall, New York, 1995.
427. Oivo, M., 'Incremental resource estimation with real-time feedback from measurement', Microprocessing-&-Microprogramming, 38(1-5), pp281-289, 1993.
428. Ojha, N. and J. D. McGregor, Object oriented metrics for early system characterization: a CRC card based approach. Technical report No. TR 94-107, Clemson University, 1994.
429. Ong, C. L. and W. T. Tsai, 'Class and object extraction from imperative code', Journal of Object-Oriented Programming, 6(1), pp58-60, 62-68, 1993.
430. Ott, L. M. and J. M. Bieman. 'Effects of Software Changes on Module Cohesion', in Proc. Intl. Conference on Software Maintenance. IEEE, 1992.
431. "P. GOODMAN, ""Implementing Software Metrics Programmes: A Project-Based Approach"" Eurometrics '92 Proceedings, April 1992"
432. Pai, W. C. and C. C. Wang, 'A formal approach to object-oriented software testing and complexity measurement with Z', International Journal of Computers & Applications, 20(3), pp147-158, 1998.
433. Pant, Y. R., J. M. Verner, et al. 'S/C: a software size/complexity measure', in Proc. 1st IFIP/SQI International Conference on Software Quality and Productivity (ICSQP'94). Hong Kong: 1994.
434. Pant, Y., B. Henderson-Sellers, et al., 'Generalization of Object-Oriented Components for Reuse: Measurement of Effort and Size Change', JOOP, 9(2), pp19-41, 1996.
435. Park, S., E. S. Hong, et al., 'Metrics measuring cohesion and coupling in object-oriented programs', Journal of KISS, 25(12), pp1779-1787, 1998.
436. Paul, R. A., T. L. Kunii, et al. 'Object-oriented evolutionary database design for software metrics data', in Proc. Twenty-First Annual International Computer Software and Applications Conference (COMPSAC'97). IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.
437. Paul, R., C. L. Chee, et al. 'Data models for metrics-based project management systems', in Proc. The Twentieth Annual International Computer Software and Applications Conference (COMPSAC '96). IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
438. Paul, R., Y. Shinagawa, et al. 'Object-oriented framework for metrics guided risk management', in Proc. The Twentieth Annual International Computer Software and Applications Conference (COMPSAC '96). IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
439. Paul, S. and A. Prakash, 'A query algebra for program databases', IEEE Transactions on Software Engineering, 22(3), pp202-217, 1996.
440. Pereira, C. E. and P. Darscht. 'Using object-orientation in real-time applications: an experience report or (1 application+5 approaches=1 comparison)', in Proc. Technology of Object-Oriented Languages and Systems, TOOLS 13. Prentice Hall, Hemel Hempstead, UK, 1994.
441. Perry, D. E. and G. E. Kaiser, 'Adequate testing and object-oriented programming', Journal of Object-Oriented Programming, 2(5), pp13-19, 1990.
442. Pfleeger, S. L. and J. Palmer. 'Software estimation for object-oriented systems', in Proc. International Function Point Users Group Fall Conference. San Antonio, TX: 1990.
443. Pfleeger, S., R. Jeffery, et al., 'Status Report on Software Measurement', IEEE Software, 14(2), pp33-43, 1997.
444. Philip, T. and R. Ramsundar, 'A reengineering framework for small scale software', SIGSOFT Software Engineering Notes, 20(5), pp51-55, 1995.
445. Pintado, X., The affinity browser, in Object-Oriented Software Composition, O. Nierstrasz and D. Tsichritzis, Editors, Prentice Hall: 1995.
446. Pittman, M., 'Lessons learned in managing object-oriented development', IEEE Software, 10(January), pp43-53, 1993.
447. Poels, G. and G. Dedene. 'Formal measurement in object-oriented software', in Proc. Object Technology '96. Christchurch College Oxford: 1996.
448. Poels, G. 'Towards a size measurement framework for object-oriented specifications', in Proc. European Software Measurement Conference. FESMA 98. Technologisch Instituut Vzw, Antwerpen, Germany, 1998.
449. Ponder, C. and B. Bush, 'Polymorphism considered harmful', ACM SIGPLAN Notices, 27(6), pp76-79, 1992.
450. Ponder, C. and B. Bush, 'Polymorphism considered harmful', ACM SIGSOFT Software Engineering Notes, 19(3), pp35-37, 1994.
451. "Poulin, J. S. and D. D. Brown. 'Measurement-driven quality improvement in the MVS/ESA operating system', in Proc. Proceedings of the Second International Software Metrics Symposium (Cat. No.94TH06478). IEEE Comput. Soc. Press, Los Alamitos, CA, USA; 1994; ix+106 pp. 1994."""
452. Poulin, J. S., 'Fueling software reuse with metrics', Object Magazine, 7(7), pp42-46, 1997.
453. Price, M. W. and S. A. Demurjian Sr. 'Analyzing and measuring reusability in object-oriented designs', in Proc. OOPSLA 97. Atlanta, GA, USA: ACM, 1997.
454. Pritchett, W. W., 'Applying object-oriented metrics to Ada 95', Ada Letters, 16(5), pp48-58, 1996.
455. Proceedings: CompEuro, March 1993
456. Qui, F., R. Foley, et al. 'A model for assessment module selection in product certification', in Proc. Software Quality Management International Conference. Edinburgh: Computational Mechanics Publications, 1994.
457. "R. FICHMAN AND C. KERMERER, ""Object-Oriented and Conventional Analysis and Design Methodologies: Comparison

and Critique"""
458. "R. KOLEWE, ""Metrics in Object-Oriented Design and Programming"" Software Development, Vol. 1, No. 4,October 1993"
459. Rains, E., 'Function points in an Ada object-oriented design?', OOPS Messenger, 2(4), pp23-25, 1991.
460. Rajaraman, C. and M. R. Lyu. 'A study of coupling in C++ programs', in Proc. Technology of Object-Oriented Languages and Systems (TOOLS8). Prentice Hall International, 1993.
461. Rajaraman, C. and M. R. Lyu. 'Reliability and maintainability related software coupling metrics in C++ programs', in Proc. 3rd Intl. Symposium on Software Reliability Engineering. IEEE Computer Press, 1992.
462. Rajaraman, C. and M. R. Lyu. 'Some coupling measures for C++ programs', in Proc. 8th Intl. Conf. Technology of Object-Oriented Languages and Systems. Santa Barbara: Prentice Hall, 1992.
463. Ramakrishnan, S. and T. Menzies. 'An ongoing OO software engineering measurement experiment', in Proc. 1996 International Conference Software Engineering: Education and Practice. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
464. Rawlings, R. 'Some numbers from two object-oriented developments', in Proc. IEE Colloquium on Object Oriented Development. London: IEE, 1993.
465. Reinold, K. 'Processes and metrics for object-oriented software development', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
466. Reyes, L. and D. Carver. 'Predicting object reuse using metrics', in Proc. SEKE '98. Tenth International Conference on Software Engineering and Knowledge Engineering. Knowledge Syst. Inst, Skokie, IL, USA: 1998.
467. Riel, A. J. 'Introduction to object-oriented design through real world examples', in Proc. OOP '94/C++ World. SIGS Publications, New York, NY, USA, 1994.
468. Rising, L. 'An information hiding metric', in Proc. OOPSLA `93 Workshop on Processes and Metrics for Object Oriented Software Development. Washington DC: 1993.
469. Rising, L. and F. W. Calliss. 'An experiment investigating the effect of information hiding on maintainability', in Proc. Phoenix Conference on Computers and Communication. 1993.
470. Rising, L. S. and F. W. Calliss, 'An information-hiding metric', J. Systems and Software, (26), pp211-220, 1994.
471. Rising, L., OO Design Heuristics. Internal Company Document , AG Communication Systems, 1994.
472. "Roberto Meli, ""FUNCTIONAL AND TECHNICAL SOFTWARE MEASUREMENT: CONFLICT OR INTEGRATION ?"", Data Processing Organization, http://web.tin.it/dpo"
473. "Roberto Meli, Luca Santillo, ""Function Point Estimation Methods: a Comparative Overview"", Data Processing Organization, http://web.tin.it/dpo"
474. Roberts, T. 'Workshop report - metrics for object-oriented software development', in Proc. OOPSLA'92. ACM, 1992.
475. Rocacher, D., Metrics definitions for Smalltalk. No. Project 1257, MUSE, Workpackage WP9A, ESPRIT, 1988.
476. Rocacher, D., Smalltalk Measure Analysis Manual. No. Project 1257, MUSE, Workpackage WP9A, ESPRIT, 1989.
477. Rosenberg, L. H. and S. B. Sheppard. 'Metrics in software process assessment, quality assurance and risk assessment', in Proc. 2nd Intl. Software Metrics Symposium. London, England: IEEE Computer Society Press, Los Alamitos, CA, USA, 1994.
478. "ROSENBERG, LINDA AND LAWRENCE HYATT, ""Software Quality Metrics for Object-Oriented System environments"" SATC, NASA Technical Report SATC-TR-95-1001, 1995."
479. Rossi, M. and S. Brinkkemper. 'Metrics in method engineering', in Proc. Advanced Information Systems Engineering. Proceedings of the 7th International Conference CAiSE*95. 1995.
480. Rubin, H. A. 'Software process maturity: measuring its impact on productivity and quality', in Proc. First International Software Metrics Symposium. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1993.
481. Rubin, H., E. Youdon, et al., Industry Canada Worldwide Benchmark Project. Survey Report , Rubin Systems Inc., 1995.
482. Rubin, K. S. 'Advanced project management', in Proc. OOP'97. SIGS Conferences, Bergisch Gladbach, Germany, 1997.
483. "S.A. WHITMIRE, ""Object-Oriented Measurement of Software"" The Encyclopedia of Software Engineering, Volume 2, J.J. Marciniak 1994"
484. "S.D. CONTE, H.E. SUNSMORE, AND V.Y. SHEN, ""Software Engineering Metrics and Models"", Benjamin/Cummings, Menlo Park, California, 1986."
485. "S.L. PFLEEGER AND J.D. PALMER, ""Software Estimation for Object Oriented Systems"" Fall International Function Point Users Group Conference, San Antonio, Texas, 1990"
486. "S.R. CHIDAMBER AND C.F. KEMERER, ""A Metrics Suite for Object-Oriented Design"" IEEE Transactions on Software Engineering, Vol. 20, No. 6, June 1994"
487. Samadzadeh, M. H. and S. J. Khan. 'Stability, coupling, and cohesion of object-oriented software systems', in Proc. 22nd Annual 1994 ACM Computer Science Conference. ACM, New York, NY, USA, 1994.
488. Samaraweera, L. G. and R. Harrison, 'Evaluation of the functional and object-oriented programming paradigms: a replicated experiment', Software Engineering Notes, 23(4), pp38-43, 1998.
489. Sarirete, A. and J. Vaucher. 'Similarity measures for the object model', in Proc. Object-Oriented Technology, ECOOP'97 Workshops Proceedings. Springer-Verlag, Berlin, Germany, 1998.
490. Schach, S. R., 'The Cohesion and Coupling of Objects', JOOP, 8(1), pp48-50, 1996.
491. Schwanke, R. W. 'An intelligent tool for re-engineering software modularity', in Proc. Proceedings of the 13th International Conference on Software Engineering (ICSE '13). IEEE, 1991.
492. Shan, Y. P., 'Smalltalk on the rise', Commun. ACM, 38(10), pp103-104, 1995.
493. Sharble, R. C. and S. S. Cohen, 'The Object-Oriented Brewery: A Comparison of Two Object-Oriented Development Methods', ACM SIGSOFT Software Engineering Notes, 18(2), pp60-73, 1993.
494. Sheetz, S. D., D. P. Tegarden, et al. 'Measuring object-oriented system complexity', in Proc. Workshop on Information Technologies and Systems WITS'91. 1991.
495. Sheetz, S. D., D. P. Tegarden, et al., 'A group support systems approach to cognitive mapping', Journal of Management Information Systems, 11(1), pp31-57, 1994.
496. Shepperd, M. J. and D. C. Ince, 'A critique of three metrics', J. Systems and Software, 26(3), pp197-210, 1994.
497. Sherif, J. S. and P. Sanderson, 'Metrics for object-oriented software projects', Journal of Systems and Software, 44(2), pp147-154, 1998.
498. Shih, T. K., C. C. Wang, et al., 'Using Z to specify object-oriented software complexity measures', Information and Software Technology, 39(8), pp515-529, 1997.
499. Shih, T. K., C. M. Chung, et al. 'Decomposition of Inheritance Hierarchy DAGS for Object-Oriented Software Metrics', in Proc. International Conference and Workshop on Engineering of Computer Based Systems. Monterey, CA, USA: IEEE Computer Society Press, 1997.
500. Shih, T. K., C. M. Chung, et al. 'Decomposition of multiple inheritance DAGs for object-oriented software measurement', in Proc. Software Quality Engineering. Comput. Mech. Publications, Southampton, UK, 1997.

501. Shih, T. K., Y. C. Lin, et al., 'An object-oriented design complexity metric based on inheritance relationships', International Journal of Software Engineering and Knowledge Engineering, 8(4), pp541-566, 1998.
502. Siebellehner, J., G. Kappel, et al., Coupling and cohesion metrics for object-oriented software development, in Theorie und Praxis der Softwaremessung, R. Dumke and H. Zuse, Editors, Deutscher Universitaetsverlag, Wiesbaden, Germany: 1994.
503. Simoens, R. 'The introduction of an object oriented analysis/design method and object oriented metrics in the software development life-cycle', in Proc. Ada in Europe. Second International Eurospace - Ada Europe Symposium. Springer-Verlag, Berlin, Germany, 1996.
504. Sneed, H. 'Applying size, complexity and quality metrics to an object-oriented application', in Proc. 10th European Software Control & Metrics Conference. Herstmonceux, England: Shaker Publishing, 1999.
505. Sneed, H. M. 'Estimating the Development Costs of Object-Oriented Software', in Proc. Evolving Systems. Durham '95. 9th European Workshop on Software Maintenance. Durham, UK: DSM Ltd, 1995.
506. Stalhane, T. 'Development of a model for reusability assessment', in Proc. Objective Software Quality. Objective Quality: Second Symposium on Software Quality Techniques and Acquisition Criteria. Springer-Verlag, Berlin, Germany, 1995.
507. Stark, M., 'Impacts of object-oriented technologies: seven years of SEL studies', ACM SIGPLAN Notices, 28(10), pp365-373, 1993.
508. Stark, M., 'Impacts of object-oriented technologies: seven years of SEL studies', J. Systems and Software, 23(2), pp163-169, 1993.
509. Steinmuller, U. 'Qualifying C++ Foundation Classes for use in Industrial Applications', in Proc. Technology of Object-Oriented Languages and Systems (TOOLS 7). Dortmund Germany: Prentice Hall, 1992.
510. Stiglic, B., M. Hericko, et al., 'How to evaluate object-oriented software development?', ACM SIGPLAN Notices, 30(5), pp3-10, 1995.
511. Succi, G., S. Doublait, et al., 'Reuse and reusability metrics in an object oriented paradigm', International Journal of Applied Software Technology, 1, pp3-4, 1995.
512. Sutcliffe, A. G., 'Object-oriented systems development: survey of structured methods', Information & Software Technology, 33(6), pp433-442, 1991.
513. Szabo, R. M. and T. M. Khoshgoftaar, Modelling software quality in an object oriented software system. Internal Report No. TR-CSE-95-05, Department of Computer Science and Engineering, Florida Atlantic University, 1995.
514. Szabo, R. M. and T. M. Khoshgoftaar, The detection of high risk software modules in an object-oriented system. Internal Report No. TR-CSE-95-07, Department of Computer Science and Engineering, Florida Atlantic University, 1995.
515. Szabo, R. M. and T. M. Khoshgoftaar. 'An assessment of software quality in a C++ environment', in Proc. 6th Int. Symposium on Software Reliability Engineering, ISSRE'95. Toulouse, France: 1995.
516. Taylor, D. A., Object-Oriented Information Systems: Planning and Implementation. Wiley: New York, 1992.
517. Taylor, D. A., 'Software metrics for object technology', Object Magazine, (March-April), pp22-25, 1993.
518. Tegarden, D. P. and S. D. Sheetz. 'Object-oriented system complexity: an integrated model of structure and perceptions', in Proc. OOPSLA '92 Workshop 'Metrics for Object-Oriented Software Development. 1992.
519. Tegarden, D. P., OO Measurement Bibliography. unpublished document , Virginia Polytechnic Institute and State University, 1994.
520. Tegarden, D. P., S. D. Sheetz, et al. 'The effectiveness of traditional metrics for object-oriented systems', in Proc. 25th Hawaii International Conference on System Sciences. IEEE Computer Society Press, 1992.
521. Tegarden, D. P., S. D. Sheetz, et al., 'A Software Complexity Model of Object-Oriented Systems', Decision Support Systems, 13(3-4), pp241-262, 1995.
522. "TEGARDEN, D., S. SHEETZ, AND D. MONARCHI, ""Effectiveness of Traditional Software Metrics for Object-Oriented Systems"" Proceedings: 25th Hawaii International Conference on System Sciences, January 1992"
523. Teologlou, G. 'Measuring object-oriented software with predictive object points', in Proc. 10th European Software Control & Metrics Conference. Herstmonceux, England: Shaker Publishing, 1999.
524. Teologlu, G. 'Estimating the cost of object-oriented programming', in Proc. 6th European Software Cost Measurement Conference (ESCOM). Rolduc, The Netherlands: 1995.
525. Thomas, D. 'Object-orientation and software quality assurance', in Proc. 8th Intl. Conf. on Software Engineering and its Applications. Paris-La-Defense: 1995.
526. Thomson, N., R. Johnson, et al. 'Project estimation using an adaptation of function points and use cases for OO projects', in Proc. Workshop on Pragmatic and Theoretical Directions in Object-Oriented Software Metrics, OOPSLA'94. 1994.
527. Thuy, N. N. 'Testability and unit tests in large object-oriented software', in Proc. Quality Week '92 Conference. 1992.
528. Ungar, D. 'Position paper', in Proc. OOPSLA'93 Workshop on Object-oriented Testing. 1993.
529. Unger, B. and L. Prechelt, The impact of inheritance depth on maintenance tasks -
530. Vaishnavi, V. K. and R. K. Bandi, 'Measuring Reuse', Object Magazine, 6(2), pp53-57, 1996.
531. Ververs, F. and C. Pronk. 'On the interaction between metrics and patterns', in Proc. OOIS'95. Dublin City University: 1995.
532. Ververs, F. and P. van Dalen. 'Measuring for (re)design', in Proc. Conference on Software Measurement and Management. Int. Function Point Users Grup, Westerville, OH, USA, 1996.
533. Vessey, I. and S. A. Conger, 'Requirements specification: learning object, process, and data methodologies', Commun. ACM, 37(5), pp102-113, 1994.
534. "Voas, J. M. 'Object-oriented software testability', in Proc. 3rd International Conference on Achieving Quality in Software. Chapman and Hall, London, 1996. ISBN 0-412-63900-9 "
535. "W. LI AND S. HENRY, ""Maintenance Metrics for the Object-Oriented Paradigm"" Proceedings of the First International Software Metrics Symposium, Baltimore, 1993"
536. Walker, I. J., 'Requirements of an object-oriented design method', Software Engineering Journal, 7(2), pp102-113, 1992.
537. Walsh, J. 'Preliminary defect data from the iterative development of a large C++ project', in Proc. OOPSLA '92. Vancouver, Canada: 1992.
538. Walsh, J., Software quality in an iterative object-oriented development paradigm. Interal Report , Rational, 1993.
539. Wang, C. C., T. K. Shih, et al. 'An automatic approach to object-oriented software testing and metrics for C++ inheritance hierarchies', in Proc. ICICS, 1997 International Conference on Information, Communications and Signal Processing. IEEE, New York, NY, USA, 1997.
540. Wang, C. C., W. C. Pai, et al. 'Using Z approach to object-oriented software testing and metrics for C++ inheritance hierarchies', in Proc. 13th International Conference on Computers and Their Applications. International Society for Computers and Their Applications -ISCA, Cary, NC, USA, 1998.
541. Wang, H. Y., C. M. Chung, et al. 'Object-oriented software quality through data scope complexity measurement', in Proc. 1997 IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation. IEEE, New York,

NY, USA, 1997.
542. Webster, S., 'An annotated bibliography for object-oriented analysis and design', Information & Software Technology, 36(9), pp569-582, 1994.
543. Wei, L., Applying Software Maintaining Metrics in the Software Development Life Cycle. Masters Thesis , MIT, 1992.
544. Wessale, W., D. Reifer, et al., 'Large project experiences with object-oriented methods and reuse', J. Systems and Software, 23(2), pp151-161, 1993.
545. West, M., A data interchange format for OO metrics data. , IBM Hursley, UK, 1994.
546. West, M., An investigation of C++ metrics to improve C++ project estimation. IBM internal paper , IBM, 1992.
547. West, M., 'Taking the measure of metrics', Object Expert, 1(2), pp43-45, 1996.
548. Whitmire, S. A. 'A formal object model for measurement', in Proc. OOPSLA'96 - workshop: OO product metrics. 1996.
549. Whitmire, S. A. 'Measuring complexity on object-oriented software', in Proc. 3rd International Conference on Applications of Software Measurement. La Jolla, CA: 1992.
550. Whitmire, S. A., Applying Function Points to Object-Oriented Software (chapter 13), in Software Engineering Productivity Handbook, J. Keyes, Editor, 1993.
551. Whitmire, S. A., Object-Oriented Design Measurement. John Wiley & Sons: New York, 1997.
552. Whitmire, S. A., Object-Oriented Measurement of Software, in Encyclopedia of Software Engineering, J. Marciniak, Editor, John Wiley and Sons Inc.: New York, 1994.
553. Whitty, R. W. 'Applying Process Maturity Assessment Data to Evaluate Research in Object-orient Software Metrics', in Proc. Bournemouth Metrics Workshop. Bournemouth, UK: 1996.
554. Whitty, R. W., 'Object-oriented metrics: a status report', Object Expert, (January/February), pp39-44, 1996.
555. Whitty, R. W., 'Object-oriented metrics: an annotated bibliography', ACM SIGPLAN Notices, 31(4), pp45-75, 1996.
556. Wilde, N. and R. Huitt, 'Maintenance support for object-oriented programs', IEEE Transactions on Software Engineering, 18(2), pp1038-1044, 1992.
557. Wilde, N., P. Matthews, et al., 'Maintaining object-oriented software', IEEE Software, 10(1), pp75-80, 1993.
558. Wilkie, F. G. and B. Hylands, 'Measuring Complexity in C++ Software', Software Practice and Experience, 28(5), pp513-546, 1998.
559. Williams, J. D. 'Metrics for object oriented projects', in Proc. Object EXPO Europe. SIGS Publications, New York, NY, USA, 1993.
560. Williams, J. D. 'Metrics for object oriented projects', in Proc. Software. DevCon '94. SIGS Publications, New York, NY, USA, 1994.
561. Williams, J. D. 'Metrics for object-oriented projects', in Proc. OOP'94 and C++ World. SIGS Publications, 1994.
562. "WILLIAMS, JOHN D., ""Metrics for Object-Oriented Projects"" Proceedings: ObjectExpoEuro Conference, July 1993"
563. Wirfs-Brock, R., B. Wilkerson, et al., Designing Object-Oriented Software. Prentice-Hall: 1991.
564. Wirfs-Brock, R., 'How designs differ', Report on Object Analysis and Design (ROAD), 1(4), pp51-53, 56, 1994.
565. Yamazaki, S., K. Kajihara, et al., 'Object-oriented design of telecommunications software', IEEE Software, 10(January), pp81-87, 1993.
566. Yogeesh, K. H. and J. G. Smith, 'Experiences in C++ and object-oriented design', Journal of Object-Oriented Programming, 5(7), pp23-28, 1992.
567. Yousfi, N. 'Measuring internal attributes of object-oriented software products', in Proc. TOULOSE '92. Software Engineering and Its Applications. EC2, Nanterre, France, 1992.
568. Zhuo, J., P. Oman, et al. 'Using relative complexity to allocate resources in gray-box testing of object-oriented code', in Proc. Fourth International Software Metrics Symposium. IEEE Comput. Soc, Los Alamitos, CA, USA, 1997.
569. Zhuo, J., R. Pichai, et al. 'On the validation of relative test complexity for object-oriented code', in Proc. Fifth International Software Metrics Symposium. IEEE Comput. Soc, Los Alamitos, CA, USA, 1998.
570. Zuse, H. and T. Fetcke. 'Properties of Object-Oriented Software Measures', in Proc. 7th Annual Oregon Workshop on Software Metrics. Silver Falls, Oregon: 1995.
571. Zuse, H. 'Foundations of object-oriented software measures', in Proc. 3rd International Software Metrics Symposium. IEEE Comput. Soc. Press, Los Alamitos, CA, USA, 1996.
572. Zuse, H., Foundations of the validation of object-oriented software measures, in Theorie und Praxis der Softwaremessung, H. Zuse and D. Reiner, Editors, DUV Deutsche Universitätsverlag: 1994.
573. Zweben, S. H., S. E. Edwards, et al., 'The effects of layering and encapsulation on software development cost and quality', IEEE Trans. on Software Eng., 21(3), pp200-208, 1995.