ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTA' DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA, INFORMATICA E TELECOMUNICAZIONI

TITOLO DELL' ELABORATO

“Analisi delle metriche del software orientate agli oggetti”

Elaborato in

Ingegneria del Software

Relatore

Giuseppe Bellavia

Presentata da

Marco Moscato

Correlatore

Antonio Natali

Sessione III°

Anno Accademico 2011/2012

ALMA MATER STUDIORUM – UNIVERSITA' DI BOLOGNA
SEDE DI CESENA
SECONDA FACOLTA' DI INGEGNERIA CON SEDE A CESENA
CORSO DI LAUREA IN INGEGNERIA ELETTRONICA, INFORMATICA E TELECOMUNICAZIONI

TITOLO DELL' ELABORATO

“Analisi delle metriche del software orientate agli oggetti”

Elaborato in

Ingegneria del Software

Relatore

….....

Giuseppe Bellavia

Presentata da

….....

Marco Moscato

Correlatore

Antonio Natali

Sessione III°

Anno Accademico 2011/2012

INDICE

Analisi delle metriche del software orientate agli oggetti

" You cannot control what you cannot measure " (Tom DeMarco)

" When you can measure what you are speaking about and express it in numbers you know something about it; but when you cannot express it in numbers, your knowledge is of a meagre and unsatisfactory kind "
(Lord Kelvin)

INTRODUZIONE

Questo elaborato si pone l'obiettivo di introdurre il lettore al ruolo che gli strumenti di misurazione, attualmente noti come "metriche software", ricoprono nel panorama attuale della programmazione ad oggetti.
Al fine di non rimanere in un ambito puramente teorico e, per offrire una visione più completa ed esauriente del possibile impiego pratico delle metriche, sono stati effettuati anche alcuni test di analisi metrica su di un framework specifico.
La misurazione è un'attività fondamentale in tutte le discipline dell'ingegneria, perché consente di acquisire nuove conoscenze con un meccanismo di valutazione oggettiva.
Questo concetto è applicabile anche all'ingegneria del software, anche se la misurazione del software risulta delicata e difficile, sia per la natura non materiale del software e sia per la relativa giovinezza delle pratiche di software engineering.
Nel 1982, l'ingegnere del software Tom DeMarco [DeMarco, 1982] ha affermato che "non è possibile controllare ciò che non si può misurare"; partendo da questo principio, negli ultimi decenni si è assegnato alle metodiche di controllo e di misurazione dei programmi un ruolo sempre maggiore.

Infatti, le metriche software costituiscono uno strumento utile per tenere sotto controllo le variabili di un progetto software e consistono nella quantificazione delle caratteristiche di un prodotto applicativo e nella possibilità di stimare e pianificare lo sforzo produttivo necessario per la realizzazione di progetti.
Le metriche software, ad esempio, offrono la possibilità di esaminare il grado di qualità complessivo del lavoro svolto, lo stato di avanzamento rispetto alla pianificazione prevista e la velocità con la quale viene portato avanti lo sviluppo da parte del team dei programmatori; l'elemento, quindi, su cui si basa il principio di applicazione delle metriche è il concetto di monitoraggio, che costituisce un elemento primario nella realizzazione del software, se si tiene conto del ruolo di crescente importanza che hanno assunto, anche nel progetto software, la produttività e la qualità del prodotto in termini oggettivi.
Ecco perché, quando si fa riferimento al concetto di metriche, si apre tutto uno scenario che, nella scienza dei computer, ha impegnato e continua tuttora a farlo, generazioni di teorici e di tecnici.
Tutti i ricercatori che operano nell'ambito della programmazione, hanno concorso allo sviluppo di questi strumenti di controllo, che possono diventare un ausilio importante per ogni programmatore, indipendentemente dal grado di complessità che un qualsiasi prodotto software abbia l'esigenza di raggiungere.
L'esistenza di questi strumenti di controllo, quindi, può contribuire a far maturare nello sviluppatore una maggiore consapevolezza ed un punto di vista critico verso il proprio operato.
Tale nuovo approccio nella programmazione ha dimostrato, negli ultimi decenni, la sua validità ed efficienza, e quindi, di pari passo con la crescente complessità raggiunta nelle procedure di programmazione, si è avvertita la sempre crescente esigenza di poter disporre, in parallelo, di ancora maggiori e nuove metodiche di misura e di controllo, che assicurassero il raggiungimento di un più alto grado di qualità ed affidabilità ed anche una maggiore manutenibilità e riutilizzabilità del codice prodotto.
In tale contesto, si può affermare che le metriche software abbiano ormai raggiunto un livello di complessità tale da essere considerate dei discreti indici di riferimento.

Oggi, quindi, si può affermare che qualsiasi produzione software può trovare nelle metriche software un valido strumento di analisi oggettiva dei risultati ottenuti nel corso del processo di sviluppo.

Le ricerche effettuate e documentate in questo elaborato si sono rivolte, capitolo per capitolo, ad una indagine che abbracciasse quelli che possono essere considerati come gli aspetti caratterizzanti delle metriche software.

In particolare:

- Nel primo capitolo vengono analizzate le origini e cioè quali siano state le cause che hanno portato alla nascita e allo sviluppo delle metriche software: partendo, quindi, dalla definizione di "metrica", la trattazione si è sviluppata verso l'evidenziazione di quelle che possono essere state le esigenze che hanno sollecitato la creazione di tale strumento di misura applicato in modo sempre più mirato alla reale esigenza del programmatore.
- Nel secondo capitolo si è passati ad identificare l'evoluzione delle metriche verso uno standard unificato: di pari passo con tale processo evolutivo, è sorta sempre più pressante l'esigenza di convergere verso un complesso di parametri di riferimento; tali parametri dovevano risultare soddisfacenti e validi per tutta una serie di esigenze comuni. Per questa finalità si è giunti ad identificare la funzione specifica e necessaria di una supervisione da parte di enti superiori di controllo, che definissero protocolli di verifica tutti orientati alla definizione di uno standard unificato.
- Il terzo capitolo delinea le prime classificazioni legate al mondo della programmazione ad oggetti; in tale ambito si è sviluppata, infatti, tutta una categoria di linguaggi affiancata ad una sempre più generale consapevolezza della loro sempre crescente validità e della estensione del loro campo di applicazione. Tutta questa serie di fattori hanno necessariamente richiesto la creazione di metriche mirate ed adeguate alla nuova situazione.
- Il quarto capitolo pone particolare enfasi e attenzione alle metriche create da Chidamber e Kemerer e note, nell'ambiente, come Metriche CK, una particolare tipologia di metriche object-oriented con riferimenti al loro utilizzo ed alle specificità relative.

- Nel quinto capitolo si passa ad un breve esemplificazione delle modalità di rappresentazione grafica e di corretta interpretazione di un processo di analisi metrica.
- Nel sesto capitolo vengono descritti i tool software maggiormente utilizzati per l'analisi dei prodotti software .
- Infine, nel settimo si propongono alcune serie di test e di utilizzo pratico delle metriche software su un framework specifico a riprova della validità di tali strumenti; in particolare si è scelto di adottare il tool NDepend, al fine di dare una chiara ed efficace panoramica delle possibilità offerte da tale strumento, in applicazione al procedimento di sviluppo di un reale prodotto software.

CAPITOLO 1

Fondamenti e motivazioni della misurazione

La *misurazione* è un'attività che fondamenta l'intera esistenza dell'uomo: è essenziale per stimare, verificare, consuntivare, controllare e supportare decisioni nella nostra quotidianità: ad esempio, in fisica, per la creazione di modelli del reale, in economia, per determinare il prezzo e le variazioni di prezzo di beni, in medicina, per avere diagnosi accurate, in ingegneria, per progettare e valutare artefatti.

Quelli elencati sono solo alcuni degli esempi riguardanti la continua necessità e ricerca di misurazione, che, tecnicamente, potremmo definire come " il processo di assegnazione di simboli, normalmente numeri, per rappresentare un attributo dell'entità di interesse, secondo regole definite " o ancora " l'insieme delle operazioni aventi per oggetto la determinazione del valore di una misura ".

Emergono dunque quattro elementi costituenti alla base di tali definizioni: l'**entità**, ovvero l'oggetto che si vuole sottoporre a misurazione, l'**attributo** dell'entità, l'aspetto di tale oggetto che interessa descrivere, la **forma** usata per rappresentare l'attributo, normalmente numerica e, infine, ma non ultimo in ordine di importanza, l'uso di **regole** per arrivare a determinare il valore dell'attributo, ossia la **misura**, attraverso un processo "ripetibile", rendendo le valutazioni il più possibile precise e riducendone il livello di soggettività.

Il quesito di maggior importanza è: " perchè misurare? ". Misuriamo qualcosa per soddisfare quesiti di interesse.

La sete di conoscenza è il motore, la causa e la chiave per poter comprendere e dominare una realtà di interesse e il *measurement* ha come fine ultimo quello di collezionare e restituire informazioni ormai essenziali all'attività di *decision making.*

Per poter prendere qualsiasi tipo di decisione, infatti è ormai sempre più essenziale il poter disporre di valori di riferimento e/o di elaborazioni della misura disponibile.

Per questo è importante introdurre il concetto di **metrica**, definibile come "la misura quantitativa del grado di possesso di uno specifico attributo da parte di un sistema, di un componente o di un processo".

Caratteristiche fondamentali per l'attività decisionale riferite al *measurement* sono:

- **appropriatezza:** la misura o metrica scelta deve essere appropriata per quella situazione;
- **quantità**: si deve disporre di un ammontare sufficiente di dati da elaborare;
- **accuratezza**: la rilevazione deve effettuarsi in modo opportuno, utilizzando un intervallo di confidenza tarato sui propri dati storici;
- **controllo**: è necessario effettuare con una certa frequenza controlli per verificare l'efficacia e l'efficienza del sistema di misurazione;
- **specifiche**: vanno definiti il contesto e gli oggetti ai quali si applica la misurazione e il livello di dettaglio;
- **standardizzazione**: è opportuno stabilire degli standard per condividere l'uso delle informazioni nel *tempo* e nello *spazio*;
- **linguaggio**: è opportuno utilizzare un linguaggio che permetta la condivisione dell'informazione tra le diverse professionalità presenti all'interno dell'impresa.

Per ottenere un maggior livello di approfondimento è opportuno introdurre il concetto di **scala di valori**, ovvero " l'insieme dei numeri o dei simboli da assegnare ad un attributo di un'entità e delle relazioni tra tali numeri o simboli ".

Le scale di valori si diversificano in funzione del tipo di operazioni che si rendono necessarie e la classificazione prevede cinque tipologie principali ordinate in senso crescente di precisione:

- **Nominale**: la relazione tra i valori consente una semplice classificazione degli oggetti della misurazione, ma non ci indica nulla sulla relazione tra essi;
- **Ordinale**: si introduce una relazione d'ordine fra gli oggetti, per cui si è in grado di determinare le posizioni relative degli oggetti (cioè dire se uno venga prima di un altro);

- **Intervalli**: si è in grado di misurare la *distanza* fra gli oggetti del dominio, e non solo posizionarli tra di loro come con le scale ordinali;
- **Ratio**: introduce l'elemento *zero* che rappresenta l'assenza totale dell'attributo che si sta misurando nell'entità sottoposta a misurazione.

  L'introduzione dello zero, inoltre, conferisce al valore di un attributo il senso di positività, negatività o nullità;
- **Assoluta**: esiste una strategia di conteggio per il quale possiamo assegnare ad ogni oggetto un *numero* univocamente determinato.

  La scala assoluta è usata per quegli attributi di un oggetto che richiedono un semplice **conteggio** di elementi.

TIPO DI SCALA	TRASFORMAZIONI AMMISSIBILI	OPERAZIONI EMPIRICHE DI BASE	ALCUNI INDICI STATISTICI APPROPRIATI	TEST STATISTICI APPROPRIATI	ESEMPI
NOMINALE	qualsiasi trasformazione uno-a-uno	determinazione di uguaglianza	Moda Frequenza	non parametrico	etichettare classificare entità
ORDINALE	$M(x) \geq M(Y)$ implica che $M'(x) > M'(Y)$ [*funzione crescente strettamente monotona*]	come la precedente, più la determinazione di "maggiore di" e "minore di"	Mediana Percentili Spearman *r* Kendall W Kendall T	non parametrico	preferenza ordinamento di entità
INTERVALLARE	$M'=aM+n(a>0)$ [*trasformata lineare positiva*]	come la precedente, più addizioni e sottrazioni	Media Aritmetica Deviazione standard Correlazione Pearson Correlazione multipla	non parametrico	temperature in gradi Fahrenheit o Celsius date di calendario orario
RATIO (DI RAPPORTI)	$M'=aM\ (a>0)$ [*trasformazione di similarità*]	come la precedente, più moltiplicazioni e divisioni	Media Geometrica Media Armonica Coefficiente di variazione Variazione percentuale Indice di correlazione	non parametrico e parametrico	intervalli di tempo temperature Kelvin lunghezze
ASSOLUTA	$M'=M$ [*identità*]				conteggio di entità

CAPITOLO 2

La metrologia applicata all'Ingegneria del Software

L'impiego di tecniche di misurazione specifiche nel campo dell'Ingegneria Software è recente tanto quanto il termine stesso, coniato ormai più di 40 anni fa in occasione delle due conferenze NATO di Garmish e Roma del 1968 e del 1969.
In tale campo il measurement ha avuto, fin dal principio, una tale importanza da portare alla nascita di una vera e proprio sottoarea di studi denominata **Software Measurement** o meglio Software Engineering Measurement, che ancora oggi necessita di ulteriori e continui approfondimenti ed offre ancora molte possibilità inesplorate.
Il software e la misurazione dello stesso sono due temi così attuali e in perenne e costante evoluzione e sviluppo, che solo negli ultimi anni si è registrata la nascita di progetti solidi, concreti e appoggiati dalla comunità di riferimento.
Tra questi, due trattano la costruzione di un *body of knowledge* per il Software Engineering (e di conseguenza anche per il Software Measurement), in maniera similare a quanto già esistente per altri domini di conoscenza, come il Project Management; tali progetti sono:

- **SWE-BOK** (**S**oft**W**are **E**ngineering **B**ook **O**f **K**nowledge), creato nel 1999 in ambito SEI (Software Engineering Institute);
- **SWEBOK** (**S**oft**W**are **E**ngineering **B**ook **O**f **K**nowledge), prodotto inizialmente dallo sforzo congiunto di IEEE e ACM, le due principali associazioni mondiali per l'informatica, ed ora dal solo IEEE.

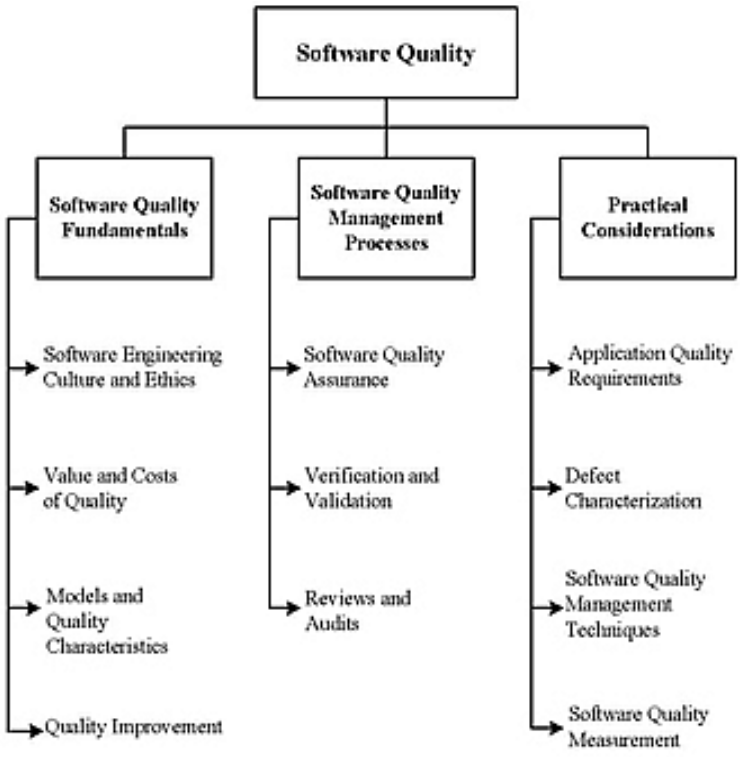
In entrambi i casi le finalità sono:

- **industriali**: valutazione delle competenze interne;
- **accademiche**: guida per la stesura di curriculum di studio relativi al Software Engineering;

- **professionali**: definizioni di uno standard per le competenze degli ingegneri del software.

Il progetto del SEI struttura la conoscenza relativa all'ingegneria del software in un BOK composto da tre livelli:

- **KC – Knowledge Category**: intesa come sottodisciplina; ogni KC c comprende una o più KA;
- **KA – Knowledge Area**: le KC vengono suddivise in aree tematiche; ogni KA è composta da una o più KU;
- **KU – Knowledge Unit**: componente elementare della conoscenza relativa al Software Engineering.



La Knowledge Category **Software Quality** è dedicata interamente al controllo ed alla misurazione della qualità del software, è suddivisa in tre Knowledge Areas distinte, tra cui quella che prende il nome di **Software Quality Management Processes** e, nello specifico, la Knowledge Unit di riferimento per le metriche d'utilizzo prende il nome di **Software Quality Measurement**.

A differenza del primo progetto, il secondo è divenuto nel 2004 un technical report ISO nell'ambito della System & Software Engineering con una struttura ed un'organizzazione che differisce per numero e categorie dal primo progetto.

L'evoluzione storica e gli obiettivi d'uso del Software: Measurement

Le caratteristiche su cui bisogna porre maggiore attenzione in materia di Software Measurement sono:

- l'*immaterialità* dell'oggetto software
- la *pervasività dell'informatica* in ogni attività moderna; questo rende necessario considerare il software quale *asset* aziendale secondo un'ottica di business, cioè come un capitale da valutare e valorizzare sia nelle aziende ICT che in quelle in cui il core business è rappresentato dall'informatica.

A partire dagli anni '70 l'attenzione è stata rivolta alle misure del codice sorgente, caratterizzate dalle Linee di Codice (LOC – Lines of Code), in un panorama di crescente accettazione della programmazione strutturata e dell'utilizzo interdisciplinare delle nozioni di *complessità cognitiva.*

Negli anni '80 ci si è concentrato sullo studio delle fasi alte del ciclo di vita del software (SLC – Software Life Cycle), applicando per la prima volta criteri di misura per la stima dei costi e dell'effort di progetto, di disegno, di specificazione e di metodi per orientare le misurazioni agli obiettivi aziendali.

Negli anni '90, infine, l'enfasi viene posta sulle misurazioni di processo, in praticolare sull'utilizzo di framework di Software Process Improvement (SPI), sulla stesura di piani metrici completi, benchmarking e su un forte interesse per la creazione di standard internazionali in tema di software measurement.

I molteplici obiettivi ed aspetti del Software Measurement sono riassumibili in quattro motivazioni principali:

- **caratterizzazione**, per conoscere le entità e l'ambiente e stabilire delle *baseline* per confronti con future valutazioni;

- **valutazione**, per determinare lo status dell'entità/attributo rispetto al pianificato;
- **previsione**, per estrapolare il trend e stimare quindi costi, tempo e qualità delle entità da produrre;
- **miglioramento**, per trasformare gli errori di percorso e le opportunità di successo in termini di qualità di prodotto e performance di processo.

Nel processo di measurement non solo è importante *definire la misura* o la metrica in funzione degli obiettivi aziendali, ma si rende necessario, nel campo del Software Engineering, effettuare una classificazione delle entità e dei relativi attributi presi in esame.

Le entità si suddividono in:

- **Processi**: la serie delle attività necessarie per la produzione del software;
- **Prodotti**: il risultato del processo di produzione;
- **Risorse**: quanto impiegato dalle attività di processo.

Gli attributi sono le qualità, le proprietà, i requisiti e le informazioni sensibili che caratterizzano le entità di interesse e che motivano le nostre misurazioni e, per questo, non possiamo esimerci dal definire due categorie distinte di attributi permettendo un ulteriore livello di profondità e precisione:

- **Attributi Esterni**: attributi di un'entità che sono visibili e di interesse per l'utente del prodotto software; descrivono l'*aspetto esterno* di un'entità e il loro rapporto con l'ambiente in cui vengono usate, indipendentemente dalla implementazione; un esempio sono la *facilità d'utilizzo*, la *portabilità*, l'*efficienza* e l'*affidabilità*
- **Attributi Interni**: attributi di un'entità che sono visibili e di interesse del produttore, i cui valori dipendono dalla implementazione; un esempio sono la *modularità*, la *strutturazione*, la *tracciabilità*, la *testabilità*, la *dimensione* e la *complessità*

	ENTITA	ATTRIBUTI INTERNI	ATTRIBUTI ESTERNI
Prodotti	Specifica	Dimensione, riuso, modularizzazione, funzionalità, correttezza sintattica	Comprensibilità, manutenibilità
	Progetto	Dimensione, riuso, accoppiamento, coesione, modularizzazione, funzionalità	Qualità, comprensibilità, manutenibilità
	Codice	Dimensione, riuso, accoppiamento, modularizzazione, funzionalità, complessità algoritmica	Affidabilità, manutenibilità, usabilità
	Dati test	Dimensione, copertura	Qualità, affidabilità
Processi	Specifica	Durata, sforzo, n. cambiamenti nei requisiti	Qualità, costo, stabilità
	Progettazione	Durata, sforzo, numero di errori individuati	Costo, stabilità
	Test	Durata, sforzo, numero di errori trovati	Costo, controllabilità
Risorse	Personale	Età,costo	Produttività, esperienza
	Team	Dimensione, livello di comunicazione, strutturazione	Produttività
	Software	Costo, dimensione	Affidabilità
	Hardware	Costo, prestazioni	Affidabilità

L'esigenza di una standardizzazione: la norma ISO/IEC 15939 e 14598

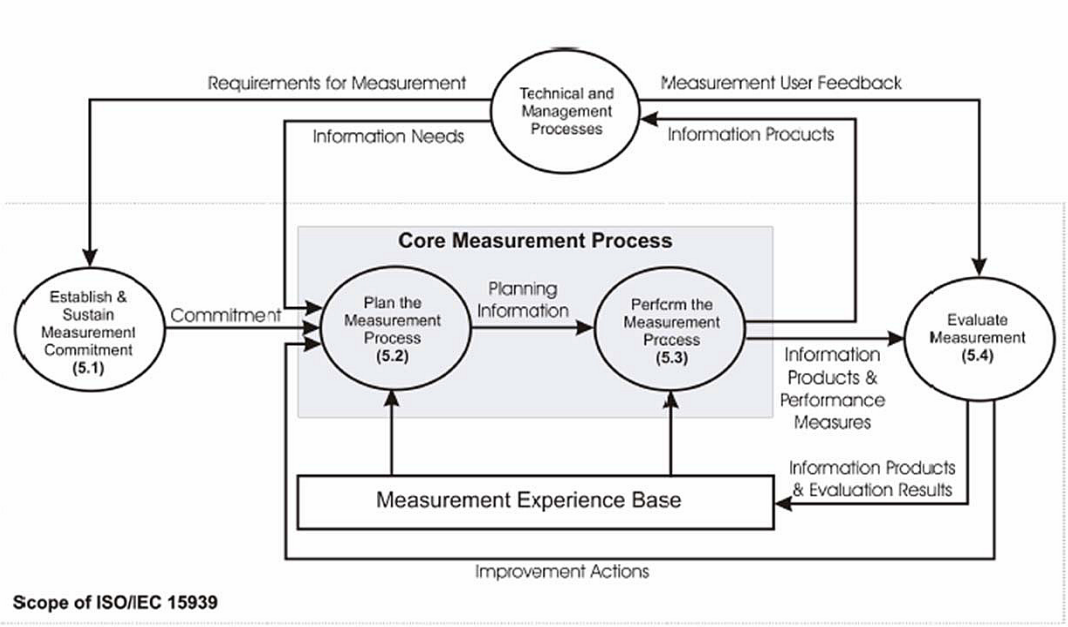
In risposta alla crescente esigenza della definizione di parametri qualitativi standard ai quali far corrispondere i processi della misurazione del software, nel 2002 si è finalmente stabilita la norma di riferimento ISO/IEC 15939.

La norma ISO/IEC 15939 si intitola " *Sofware Enigneering – Software measurement process* " e definisce i processi comuni e le procedure per identificare, definire, selezionare, applicare, validare e perfezionare le misurazioni software nell'ambito di un progetto o di una struttura di misurazione organizzativa globale.

Gli obiettivi posti da tale norma sono:

- Fornire una **definizione comune** per il processo di misurazione
- Stabilire le **caratteristiche** del processo di misurazione
- Stabilire una **base di raccolta e di utilizzo** dei dati di misurazione
- Definire una **terminologia** comune di misurazione

Fondamentalmente il processo di misura è descritto nella norma come un ciclo iterativo che prevede la misurazione e l'utilizzo dei feedback per impostare azioni correttive per migliorare il processo di produzione (PDCA Cycle).



I termini utilizzati dalla norma si basano sull' " *International vocabulary of basic and general terms in metrology* (ISO Dguide 99999:2004) ".

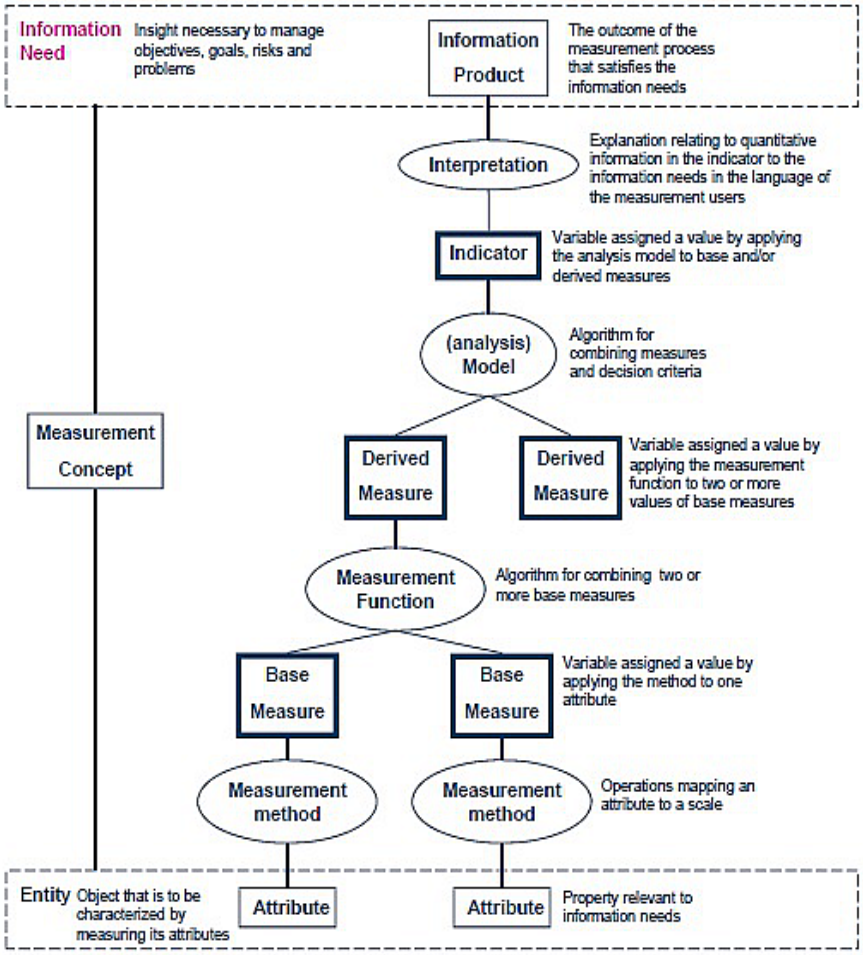
Secondo la 15939, come si può notare dalla tabella, il processo iterativo di misurazione del software è suddiviso in 4 attività principali:

1) **Definire** l'esigenza della misurazione
2) **Pianificare** la misurazione
3) **Rilevare** le misure
4) **Valutare** le misure rilevate

Nel ciclo di vita del software la misurazione serve a prevedere o stimare tempi di consegna, costo di lavorazione e qualità del prodotto, ma si può anche misurare lo stato raggiunto dal prodotto nei suoi vari stadi di lavorazione, per due obiettivi principali:

- Prevedere quali caratteristiche assumerà il prodotto software nelle fasi del relativo ciclo di vita successive a quella attuale di valutazione

- Stimare le caratteristiche possedute dal prodotto software nella fase e nello stadio di sviluppo in cui si effettua la valutazione



In totale sono tre le fasi principali del ciclo di vita di un prodotto software dove le valutazioni e le misurazioni delle caratteristiche dello stesso sono ormai essenziali:

- **Fase di progettazione**: per ottimizzare gli interventi futuri di manutenzione dopo il rilascio, valutare l'impatto ed il costo che avranno nel contesto di utilizzo e prevenire problemi nel software rilasciato in esercizio;
- **Fase di collaudo e/o test**: per confrontare il risultato del processo di sviluppo del prodotto software con le specifiche di riferimento date e scoprire eventuali problemi non considerati in fase di progettazione;
- **Fase successiva al rilascio in esercizio**: per misurare l'impatto del prodotto sull'efficienza e sull'efficacia del lavoro svolto dal fruitore del prodotto,

confrontare le prestazioni del prodotto con quelle di altri prodotti comparabili, individuare possibili aree di miglioramento e considerare il momento del ritiro della produzione di una versione obsoleta.

Questione fondamentale è stabilire che cosa del prodotto software di interesse deve essere misurato: ogni prodotto software non solo è costituito da codice sorgente, ma, ad esempio, anche da elementi supplementari come documentazione, regole, dati e specifiche.

Ognuna di queste entità inoltre, nel contesto del ciclo di vita del software, assume stadi diversi e, di conseguenza, vanno effettuate sia misure intermedie sui semilavorati, molto spesso con processi di misurazione e tempistiche differenti a seconda delle casistiche e delle tipologie, che sugli stati finali delle entità che compongono il prodotto software in oggetto.

Le misure utilizzate per valutare il software godono di alcune caratteristiche specifiche, ma sulla completezza dei risultati e l'efficienza di tali specifiche, ovviamente, molto influisce anche la metodologia di applicazione delle stesse, infatti: uno scarso rispetto formale della teoria della misura porta a stime soggette ad errori e ad incertezze, in quanto applicate ad un sistema adattivo non lineare e complesso, in cui interagiscono molte variabili, in parte legate a fattori umani, in parte organizzativi ed in parte gestionali.

La complessità e le proporzioni delle entità software che si vogliono misurare, soprattutto nel panorama attuale, portano alla distinzione di due grandi categorie di misurazioni: **Misure Interne** del Software e **Misure Esterne** del Software.

- **Misure interne del software**, per fase del ciclo di vita:
  - Misure nell'**analisi dei requisiti**: utili per le implicazioni, a cascata, sul ciclo di vita del software e per stimare il costo del software:
    - *Numero dei requisiti*
    - *Function Points* (requisiti funzionali)
    - *Volatilità dei requisiti*

- Misure nelle **specifiche**: utili per le implicazioni, a cascata, sul ciclo di vita del software e per stimare il costo del software:
  - *Numero delle entità in un diagramma E/R*
  - *Numero blocchi in un diaframma DFD*
  - *Numero degli Use Case Points (UML)*

- Misure nella **progettazione**: permettono di stimare il costo del software, ma anche alcuni aspetti di qualità:
  - *Numero di moduli in uno schema di progettazione*
  - *Livello di coesione tra moduli*
  - *Livello di accoppiamento (coupling) tra moduli*
  - *Complessità di flusso*

- Misure sul **codice**: permettono di stimare il costo del software, ma anche alcuni aspetti di qualità. Sono le più diffuse e numerose:
  - *Linee di codice (LOC o DSI)*
  - *Complessità ciclomatica*
  - *Misure di Halstead*
  - *Misure di complessità*
  - *Misure di Troy*
  - *Misure object oriented*
  - *Misure di coesione funzionale*

- Misure sul **test**: permettono di verificare la qualità del test effettuato sul software:
  - *Livello di copertura di istruzioni, rami e dei percorsi base*

➢ **Misure esterne del software**: sono tra le più diffuse, ma anche tra le meno scientificamente comprese, in quanto hanno diverse sfaccettature e potenziali interpretazioni:

- *Metriche di Gild*
- *Modello di qualità di Bohem*

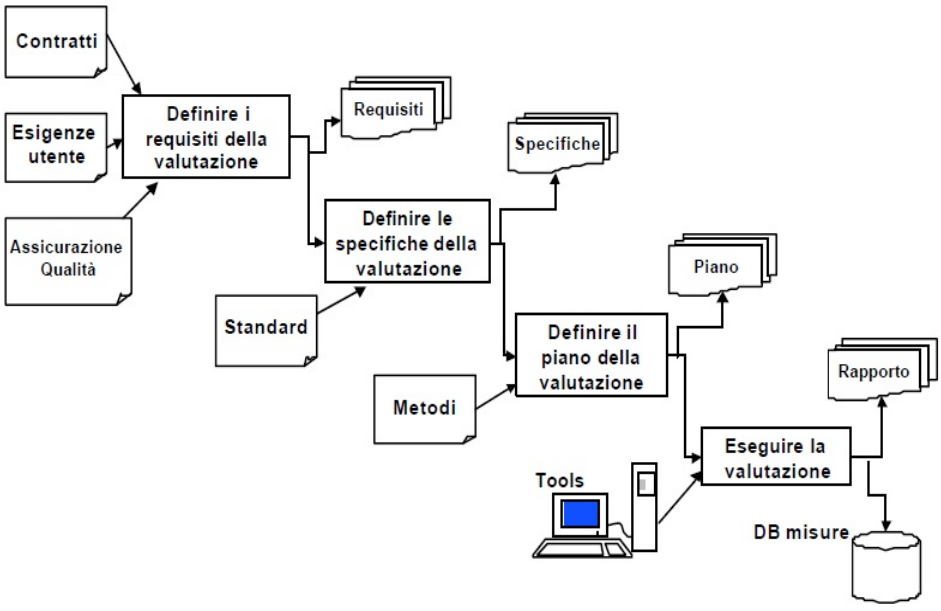
- *Modello di qualità di McCall*
- *IO/IEC 9126-1 (in pubblicazione ISO 25000)*

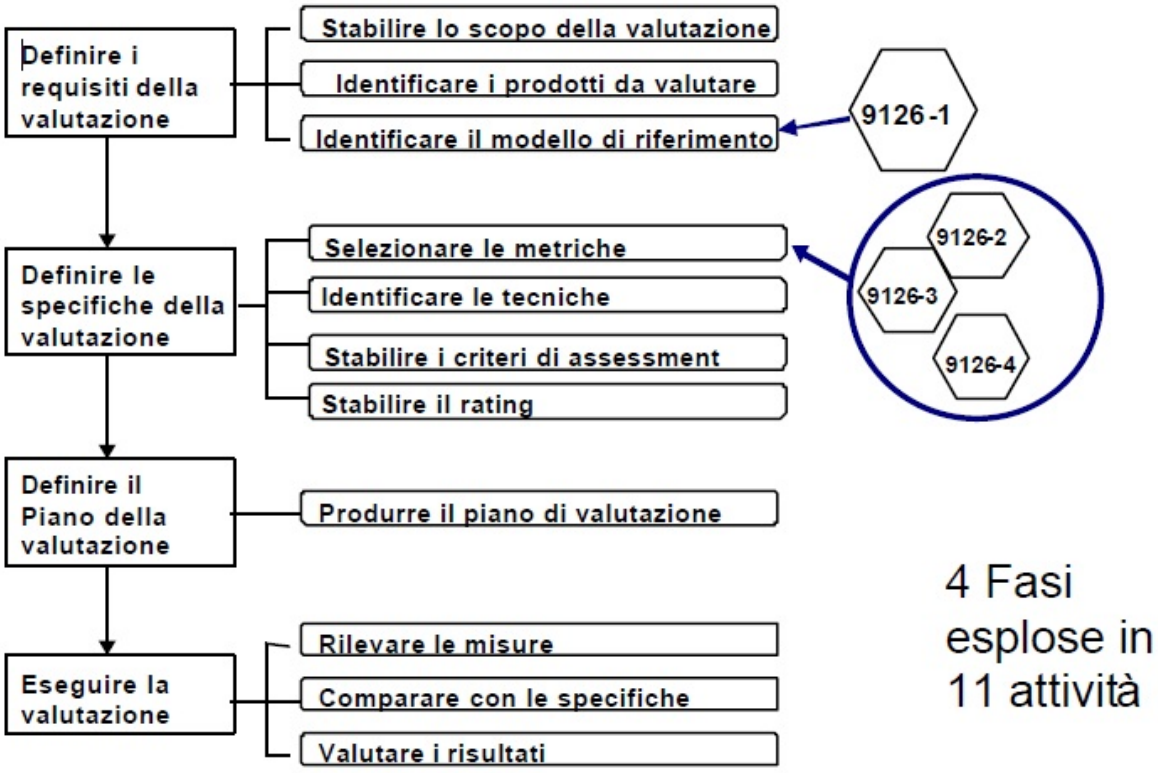
Mentre la norma ISO/IEC 15939 stabilisce quelle che sono le direttive di riferimento per il processo di misurazione delle caratteristiche di un prodotto Software, la norma ISO/IEC 14598 descrive il processo di valutazione della qualità del software in accordo con la norma ISO/IEC 9126.

Norma composta di 6 parti e intitolata " *Information Technology, Software Product Evaluation* ", propone un modello di valutazione applicabile sia ai prodotti *commercial-off-the-shelf*, sia a quelli sviluppati su commessa in base a specifici requisiti utente (*custom software* o software *ad hoc*), sia alle personalizzazioni di prodotti COTS.

Lo schema del processo di valutazione in base alle direttive di questa norma può essere riassunto in quattro fasi principali:

1) **Definizione** dei **requisiti** di valutazione
2) **Definizione** delle **specifiche** di valutazione
3) **Definizione** del **piano** di valutazione
4) **Esecuzione** della valutazione





Ogni fase principale prevede una o più sotto-fasi secondarie per un totale di undici:

Nei requisiti della valutazione vanno definiti i seguenti elementi:

- la descrizione dello scopo della valutazione
- la identificazione dei prodotti da valutare
- le attese dell'utente, in termini di livello delle caratteristiche di qualità che deve possedere o eventualmente raggiungere il prodotto software una volta terminato

La valutazione non solo può essere effettuata sul prodotto finito, ma anche, durante le fasi intermedie di produzione dello stesso, sui semilavorati.

Gli scopi della valutazione in termini di qualità del prodotto possono essere vari e numerosi, alcuni potrebbero essere:

- decidere dell'accettabilità del prodotto
- confrontare il prodotto con soluzioni della concorrenza e valutare eventuali modifiche e/o sostituzioni

- decidere non solo “se“ ma anche “quando” modificare il software per adeguarlo ad eventuali nuove esigenze, migliorarne le caratteristiche o correggere problemi
- decidere sulla rimozione o sull'accettazione di stati intermedi del prodotto
- decidere se una determinata fase sia da considerarsi completata e si possa passare alla successiva
- prevedere le qualità del prodotto finito in base a quella degli stati intermedi del prodotto
- rilevare informazioni sugli stati intermedi del prodotto al fine di migliorare la gestione del processo

La qualità del prodotto può essere una caratteristica influenzata dalla soggettività di chi deve effettuare la propria valutazione. Per questo è opportuno definire i vari ruoli in gioco nel ciclo di vita di un prodotto software; tali ruoli sono:

- L'**Acquirente**: definisce i requisiti di qualità “esterni” ed “in uso” che desidera nel prodotto e valuta le soluzioni proposte dagli sviluppatori nelle varie proposte tecniche
- Il **Fornitore**: valuta i propri prodotti affinchè corrispondano alle esigenze dell'utente e non offrano funzioni inferiori a quanto comparabile sul mercato
- Lo **Sviluppatore** di prodotti custom: deve tradurre le aspettative qualitative pretese dal Committente in specifiche tecniche e requisiti di qualità interni
- Il **Collaudatore**: si accerta della conformità delle caratteristiche del prodotto consegnato rispetto alle specifiche espresse dal Committente
- Il **Manutentore** del prodotto: deve verificare che non si siano degradate le caratteristiche qualitative del prodotto sottoposto a manutenzione dopo il rilascio in esercizio
- Il **Gestore** del prodotto: deve assicurarsi che le caratteristiche desiderate siano raggiunte e mantenute nelle condizioni operative del

prodotto, e fornire informazioni utili alla pianificazione di eventuali aggiornamenti migliorativi.

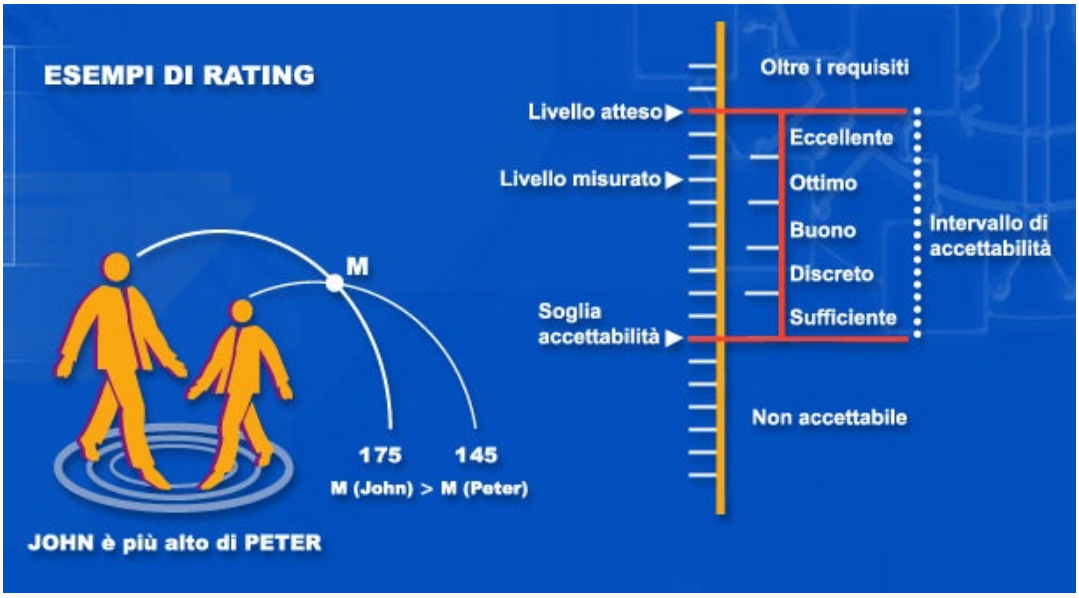
In funzione dello stato di lavorazione si possono valutare differenti caratteristiche, per esempio, nei documenti progettuali si indaga sulla correttezza e la validità formale, come la correttezza semantica, la chiarezza delle descrizioni, la comprensibilità e l'accuratezza, mentre nel prodotto in fase di collaudo ci si concentra su tutte le caratteristiche sia interne che esterne.

Per quanto riguarda l'usabilità e la qualità in uso, queste si possono riscontrare solo quando il prodotto è in esercizio, e quindi, già in mano agli utilizzatori, per questo è necessario provvedere all'intervista di un campione rappresentativo di utenti con tecniche appropriate.

Nelle specifiche di valutazione vanno sempre considerate le fonti da cui rilevare le misure necessarie alla valutazione, le tecniche che verranno usate per effettuare le valutazioni, le metriche utilizzate per rilevare le misure software e le elaborazioni che verranno effettuate sui dati recuperati.

Per misurare il livello di soddisfazione degli obiettivi posti dalle specifiche di valutazione si avvia quello che prende il nome di *processo di rating*: tale procedimento consiste, inizialmente, nell'individuare un valore “minimo” e un valore “massimo” della scala associata alla misura derivata; tali valori possono essere fatti corrispondere rispettivamente al valore più basso ed a quello più alto rilevabili dalla misura primaria e, infine, utilizzando una scala ad intervallo, si stabilisce un intervallo di valori misurabile della misura primaria, che va distribuito uniformemente nell'intervallo definito del codomio della misura derivata.

Il risultato sarà che a valori crescenti delle misure primarie dovrà corrispondere un proporzionale maggiore rispetto dei valori delle misure derivate.



La principale problematica legata al principio del rating consiste nella mancanza di fondamento "scientifico" della traduzione di una scala a rapporti o ad intervalli in una scala ordinale.

Un esempio di processo di rating:

Complessità ciclomatica	Valutazione dei rischi di modifica	Manutenibilità
1-10	Un programma senza rischi di modifica	Alta
11-20	Rischi moderati	Buona
21-50	Alti rischi	Bassa
>50	Un programma non testabile, rischi molto alti	Insufficiente

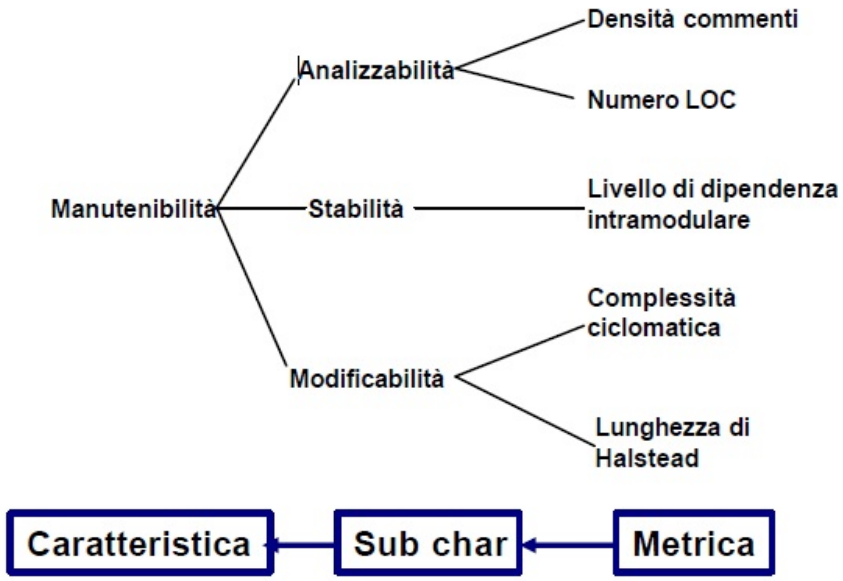
Scala ad ntervalli

Scala ordinale

In genere una determinata caratteristica è misurata applicando più metriche, e per determinare l'apporto di ogni singola metrica impiegata, si può sfruttare un albero di valutazione pesato.

Questo particolare supporto grafico permette di considerare caratteristiche, sottocaratteristiche e metriche come nodi di un albero dove le metriche rappresentano i nodi terminali (o foglie).

Ogni nodo è dotato di una determinata pesatura con un valore nell'intervallo [0,1] dell'insieme dei numeri reali, riflettendo l'importanza relativa del singolo nodo ai fini della determinazione del valore della radice.
La somma di tutti i valori relativi ai singoli nodi con un padre in comune deve essere uguale a 1.
Un esempio di struttura di albero di valutazione:



Un esempio di albero di valutazione con pesature:

Caratteristica	Sotto-carat-	peso	indicatore	peso	Val. ass.	metrica	Peso	Val. ass.
AFFIDABILITÀ	Maturità	0,2	densità di fermi in un periodo	1	0,2	n° di errori nel periodo	0,7	0,14
						n° di errori / tempo operativo	0,3	0,06
	Tolleranza	0,3	Capacità di evitare fermi in caso di errori	0,7	0,21	Numero di fermi / numero degli inconvenienti	1	0,21
			Capacità di ripristino	0,3	0,09	Durata dei fermi / n° di interruzioni del servizio	1	0,09
	Recoverabilità	0,5	Disponibilità	0,4	0,2	Tempo operativo effettivo / tempo operativo teorico	1	0,2
			Tempo di ripartenza dopo un errore	0,6	0,3	Durata massima di un femo	1	0,3

Una particolare specifica di valutazione si può considerare soddisfatta se si è raggiunto quello che si definisce il *valore di soglia*, ovvero, il valore atteso

per le misure rilevabili in riferimento al quale vengono giudicate le misure rilevate durante le verifiche.

Per fissare le soglie va considerato il fatto che, aumentando il grado qualitativo richiesto, aumenta, di conseguenza, il costo di produzione del software.

Inoltre, superata una certa soglia prestabilita, il rapporto qualità-costi di produzione non è lineare: a piccoli incrementi di qualità attesa corrispondono grandi aumenti dei costi di produzione.

Concludendo, il processo di valutazione del grado qualitativo del prodotto software può essere scomposto in tre fasi principali:

- **Measurement**: vengono rilevate le misure selezionate nella fase di pianificazione della valutazione;
- **Rating**: vengono mappati i valori rilevati su una scala di valutazione, in base a criteri stabiliti;
- **Assessment**: si confrontano gli obiettivi che si intendono raggiungere con lo stato attuale delle misure, determinando l'eventuale gap esistente.

CAPITOLO 3

Le metriche Object-Oriented: la prima classificazione

Verso la fine degli anni Ottanta, con il sempre più frequente affidamento alla programmazione orientata agli oggetti (object-oriented programming), si è sentita sempre più l'esigenza di poter usufruire di metriche maggiormente adatte a questo nuovo tipo di approccio, capaci di gestire in maniera più funzionale e precisa le caratteristiche sia interne che esterne dell'output software di nuova concezione.

L'introduzione del concetto di "oggetto" ha reso obsolete quelle metriche tradizionali che consideravano dati e funzioni come entità separate.

Grande novità ed elemento di notevole importanza è che le metriche object-oriented oltre a fornire dati, calcoli e misurazioni precise sulla struttura e il comportamento di un software OO, costituiscono uno strumento essenziale per tutto quello che la gestione del software (software management) rappresenta.

Basandosi sulle seguenti caratteristiche fondamentali della programmazione OO:

- **Modularità**
- **Ereditarietà**
- **Riutilizzabilità**
- **Incapsulamento**
- **Polimorfismo**

le metriche object-oriented focalizzano la propria attenzione su:

- la struttura interna dell'oggetto, riflettendo la complessità di ogni entità individuale
- la complessità esterna, misurando le interazioni fra entità

Inoltre si possono ottenere misurazioni precise sulla complessità computazionale che influisce sempre su:

- efficienza di un algoritmo
- efficienza d'impiego delle risorse hardware a disposizione
- fattori di carattere psicologico che, a loro volta, influiscono, in proporzioni più o meno variabili, sulle capacità creative e tecniche dello sviluppatore

Come abbiamo detto in precedenza, lo studio e la standardizzazione di queste nuove metriche è in piena fase di sviluppo e una vera e propria classificazione non è ancora stata accettata.
Una prima classificazione valida può essere considerata quella adottata da **Mark Lorenz e Jeff Kidd** nel loro libro *"Object-Oriented Software Metrics"* che vede le metriche OO divise in due particolari categorie:

- **Project Metrics** (Metriche di pianificazione): il cui scopo è quello di predire le necessità progettuali e misurare i cambiamenti dinamici che possono intercorrere nelle varie fasi previste dal progetto.
Sono metriche più generiche e meno specifiche rispetto alle metriche di progetto.
- **Design Metrics** (Metriche di Progetto): il cui scopo è la misurazione di condizioni statiche relative al design progettuale in uno specifico momento temporale.
Sono metriche di natura più localizzata e prescrittiva che pongono grande attenzione alla qualità di realizzazione del progetto.

Per ogni metrica possono essere definiti i seguenti attributi:

- **Nome**: un nome descrittivo univoco
- **Significato**: una descrizione dell'informazione che fornisce la metrica all'utente
- **Risultati di progetto**: grafici statistici dei risultati ottenuti

- **Fattori influenzanti**: fattori che, non essendo immediatamente calcolabili o evidenti possano influire sulle misurazioni ottenute dall'impiego della metrica
- **Metriche correlate**: correlazioni tra differenti metriche coinvolte nello studio progettuale
- **Livelli di riferimento**: valori euristici impiegati per stabilire soglie accettabili e inaccettabili per l'esame dei risultati ottenuti
- **Azioni suggerite**: suggerimenti dati all'utente nel caso che i valori ottenuti dalle misurazioni non rispettino i valori di soglia attesi per un risultato ottimale

La classificazione di Lorenz e Kidd prevede le metriche principali suddivise in 3 categorie principali:

- **Application Size**
- **Staffing Size**
- **Scheduling**

Le metriche appartenenti alla tipologia Application Size sono state create con la finalità di poter concepire in maniera approfondita la quantità di lavoro necessaria per una specifica applicazione.
La quantità di lavoro è una variabile difficilmente prevedibile, perchè soggetta a diversi fattori caratterizzanti l'applicazione come: il dominio di riferimento, il livello di complessità che ci si è prefissati di raggiungere e gli obiettivi principali che l'applicazione deve realizzare.
Le metriche che appartengono esattamente a questa categoria sono quattro:

1) **Number of scenario scripts (NSS)**: fornisce un'indicazione sulle dimensioni che l'applicazione potrà raggiungere al termine delle sviluppo in base ai sottosistemi, alle classi e alle suite di test previsti per la sua realizzazione.
2) **Number of key classes (NKC)**: fornisce un'indicazione sulla mole di lavoro richiesta per lo sviluppo dell'applicazione e sull'ammontare di

componenti riutilizzabili a lungo termine per future applicazioni appartenenti allo stesso dominio di utilizzo.

3) **Number of support classes (NSC)**: fornisce un'ulteriore indicazione sulla mole di lavoro prevista raffinando il calcolo in base alle classi di supporto realizzate successivamente nelle fasi avanzate di sviluppo.

4) **Number of subsystems (NOS)**: comporta un incremento nelle possibilità di scheduling.

L'architettura tra i gruppi è controllata attraverso l'uso delle interfacce dei sottosistemi, che permettono la creazione di una migliore documentazione attraverso quelle astrazioni che realizzano e consentono agli sviluppatori di prendere maggior visione di sistemi anche molto complessi.

Le metriche appartenenti alla tipologia Staffing Size consentono di focalizzarsi sull'impiego di risorse umane nella realizzazione di un determinato progetto, in questo caso le principali sono due:

1) **Person-days per class (PDC)**: fornisce un'indicazione sulla quantità di lavoro previsto per ogni sviluppatore nel conseguire determinati obiettivi di sviluppo

2) **Classes for developer (CPD)**: fornisce un'indicazione su quanti sviluppatori può arrivare a richiedere la realizzazione di un determinato componente del progetto.

Infine le metriche appartenenti all'ultima categoria Scheduling, che si riferiscono a prodotti software di tipo commerciale e che rappresentano un utile strumento di pianificazione temporale dei processi nell'ambito della gestione dei progetti, sono due:

1) **Number of major iterations (NMI)**: l'iterazione, durante lo sviluppo del software, consente, soprattutto nel campo di prodotti software sempre più complessi e lunghi nei tempi di realizzazione, di accelerarne la validazione.

2) **Number of contracts completed (NCC)**: i contratti, che rifiniscono i protocolli pubblici di servizio del sistema durante lo sviluppo, rappresentano

un utile metro di qualità nei progressi dello sviluppo delle funzionalità per l'utente finale.

Di seguito è riportata una tabella delle cross-dependency tra le metriche mostrate:

	NSS	NKC	NSC	NOS	PDC	CPD	NMI	NCC
Application Size								
Number of scenario scripts (NSS)								
Number of key classes (NKC)			◆					
Number of support classes (NSC)		◆						
Number of subsystems (NOS)								
Staffing Size								
Person-days per class (PDC)		◆				◆		
Classes per developer (CPD)					◆			
Scheduling								
Number of major iterations (NMI)								
Number of contracts completed (NCC)								

Where: ◆ = cross-dependency between metrics exists

CAPITOLO 4

Le metriche Object-Oriented: metriche CK

L'altra categoria principale alla base della classificazione di Lorenz e Kidd, che prende il nome di *Design Metrics*, racchiude metriche di progetto object-oriented che consentono di valutare il grado di complessità di un sistema software fin dalle fasi iniziali del progetto, analizzando le classi, gli oggetti e le interazioni tra di essi al fin di gestire, in modo controllato, la crescente complessità del prodotto software e, all'occorrenza, di intervenire con una riprogettazione mirata di uno o più componenti che siano risultati problematici.

Ovviamente il fine delle misurazioni derivanti dall'impiego di tali metriche è quello del raffinamento del progetto software in tutto il suo complesso, allo scopo di incrementare e ottimizzare il livello qualitativo in tutti gli aspetti, senza mai tralasciare la manutenibilità e la riusabilità del maggior numero di componenti possibili.

Prima che si manifestasse l'esigenza di metriche completamente dedicate al nuovo approccio OO, due delle metriche più comunemente utilizzate erano le seguenti:

- **Lines of Code (LOC)**
- **Cyclomatic Complexity**

La metrica *Lines o f Code* è una metrica finalizzata all'analisi del codice prodotto, appartiene alla categoria delle metriche dimensionali e, in dettaglio, misura la lunghezza di un prodotto software in termini di numero di linee di codice.

Ovviamente, quando si parla di linee di codice, bisogna specificare inizialmente quali sono le tipologie di interesse e alcune possibilità possono essere:

- qualsiasi linea di codice, inclusi anche i commenti
- tutte le linee di codice, esclusi i commenti: in questo caso è necessario il distinguo tra righe di codice effettive (LOC – ELOC) e linee di codice che non rappresentano commenti (LOC – NCLOC)
- tutte le istruzioni eseguibili (EXLOC), incluse le istruzioni dichiarative dei dati (DDLOC)

I commenti sono, troppo spesso, trascurati e sottovalutati dagli sviluppatori, o per eccessiva sicurezza del proprio operato o per la sempre più comune inadeguatezza dei tempi tecnici di realizzazione a disposizione.
Proprio tra le misure che si possono ricavare tramite l'uso della metrica LOC, una delle più importanti prende il nome di Densità di commento che consiste nel seguente rapporto:

CLOC / LOC

Una delle definizioni più accettate della metrica Lines of Code (LOC) è la seguente:

“ Una linea di codice consiste in ogni linea di testo di un programma che non sia bianca o un commento, indipendentemente dal numero di istruzioni o frammenti di istruzioni in essa contenuti. In particolare, tali linee di codice possono contenere intenstazioni di unità di programma, dichiarazioni ed istruzioni esecutive “

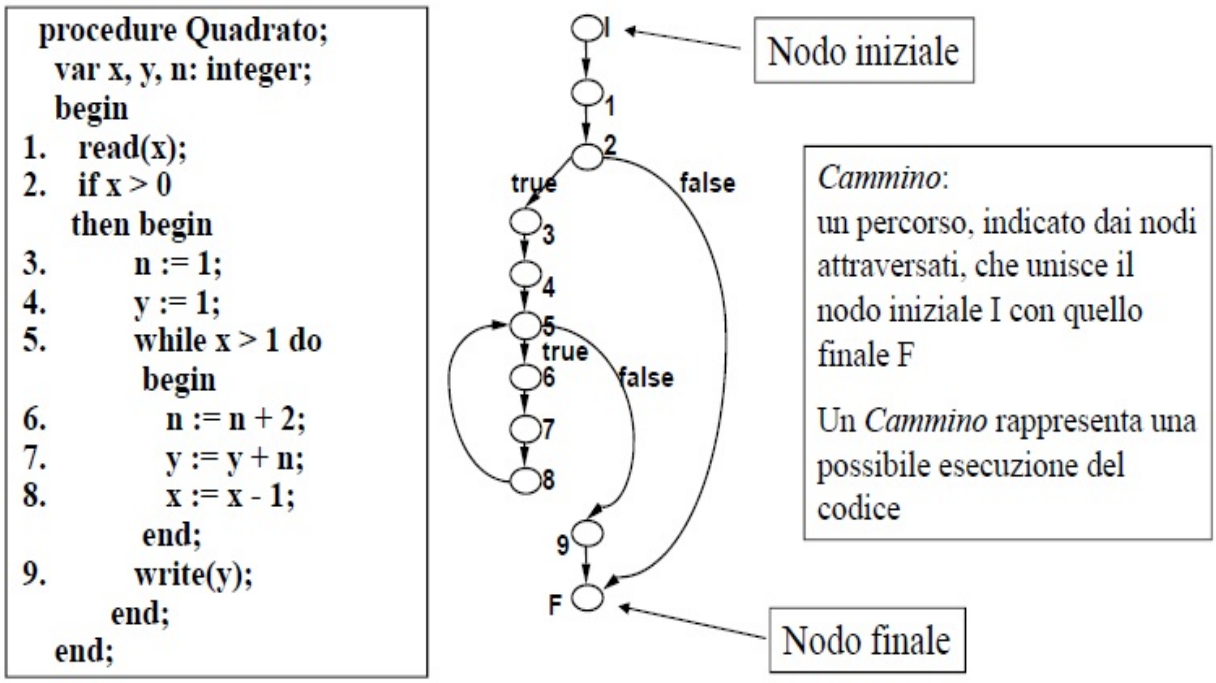
Questa metrica presenta, soprattutto scontrandosi con il mondo della programmazione ad oggetti, diverse limitazioni e problematiche, dimostrando l'inadeguatezza delle misurazioni offerte in base al nuovo approccio di programmazione, nel caso della LOC in particolare urge sottolineare le seguenti:

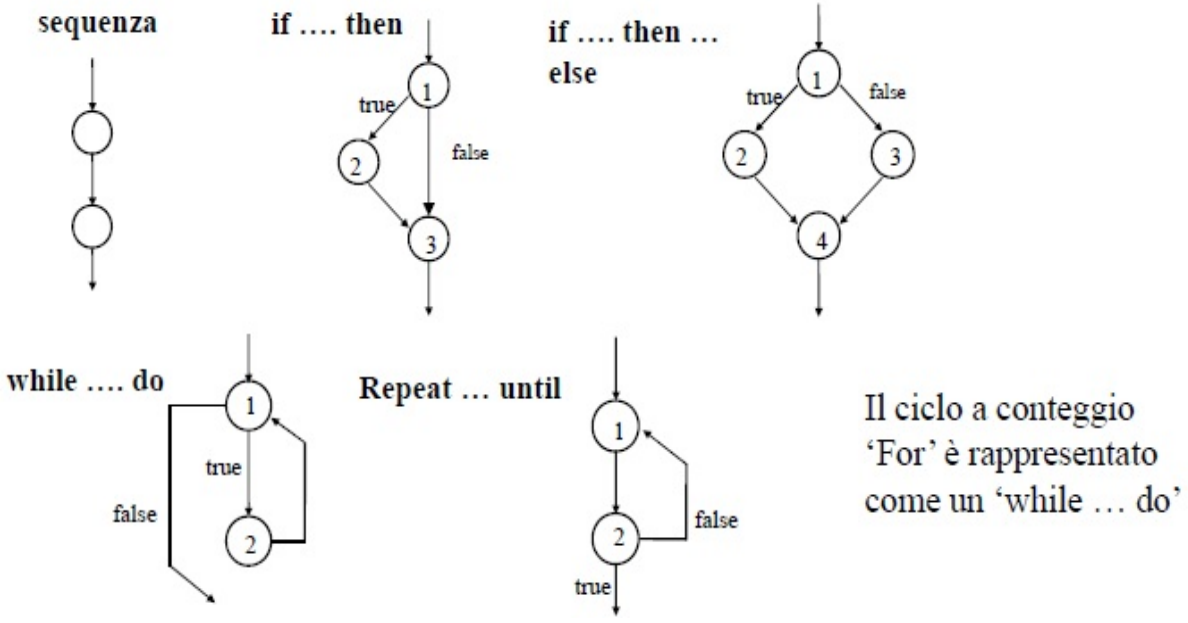
1) La LOC non fornisce risultati consistenti tra i diversi linguaggi, applicazioni e sviluppatori: non si tiene conto, infatti, delle differenze espressive dei molteplici linguaggi disponibili e della possibilità, in molti di essi, di suddividere il codice di un'istruzione su più linee
2) La complessità del codice non si riflette nella LOC
3) La LOC incoraggia un volume di codice maggiore
4) La LOC non può fornire una buona predizione sul fattore di qualità o sullo stato dei progressi del prodotto software in oggetto

Consapevoli delle limitazioni tipiche di questa particolare metrica, con l'avvento delle prime metriche object-oriented, la metrica LOC è stata maggiormente impiegata nella misurazione indiretta di altri attributi come, ad esempio:

1) La probabilità di presenza di errori nel programma
2) Il tempo necessario per completare lo sviluppo del prodotto software
3) La produttività di uno sviluppatore
4) La stima dei costi di un particolare progetto

La seconda metrica di riferimento sopracitata, la *Cyclomatic Complexity* o *Metrica di complessità di McCabe*, è una metrica software sviluppata da Thomas J. McCabe nel 1976 che si prefigge di analizzare e rappresentare il grado di complessità di un software e lo sforzo richiesto per realizzarlo.
E' una metrica di tipo strutturale relativa al flusso di controllo di un determinato programma e ne misura direttamente il numero di percorsi linearmente indipendenti presenti all'interno del codice sorgente.
Il codice viene rappresentato tramite un grafo: il grafico del flusso di controllo (Control Flow Graph), i cui nodi corrispondono a statement (istruzioni e/o predicati) del programma e gli archi al percorso di iterazione del flusso di controllo.
Di seguito un semplice codice di esempio e il suo relativo grafico del flusso di controllo:





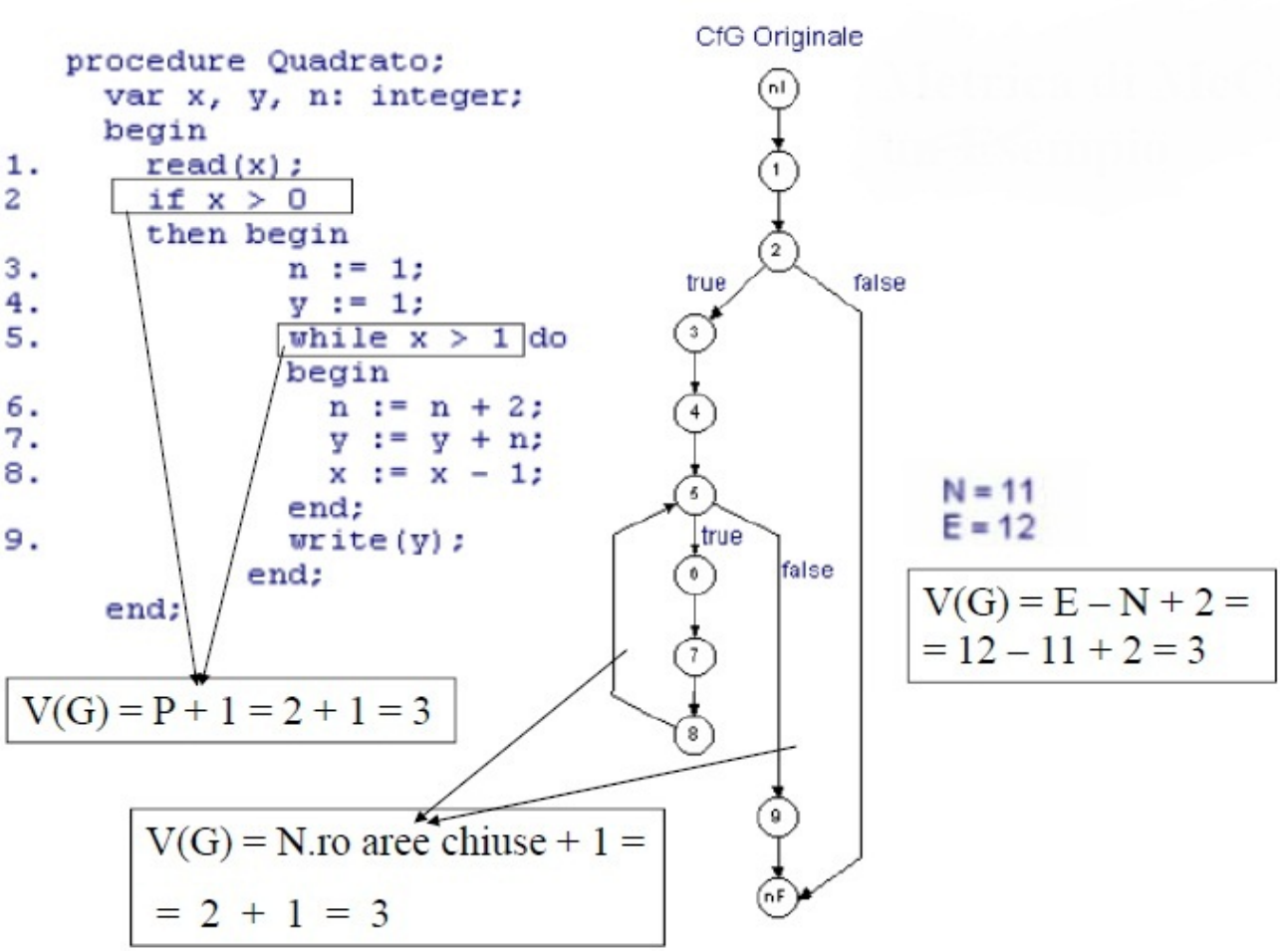
Il valore fondamentale risultante dallo studio dell'applicazione tramite l 'utilizzo del grafo del flusso di controllo prende il nome di **numero ciclomatico**, la cui formula di calcolo è la seguente:

$$V(G) = E - N + 2P$$

dove **G** rappresenta il codice sorgente del programma in analisi, **E** il numero di archi, **N** il numero di nodi e **P** il numero di componenti connesse.
Un metodo alternativo per calcolare V(G) è:

V(G) = numero di aree chiuse del CfG + 1

Il risultante numero ciclomatico dell'esempio di studio sarà il seguente:

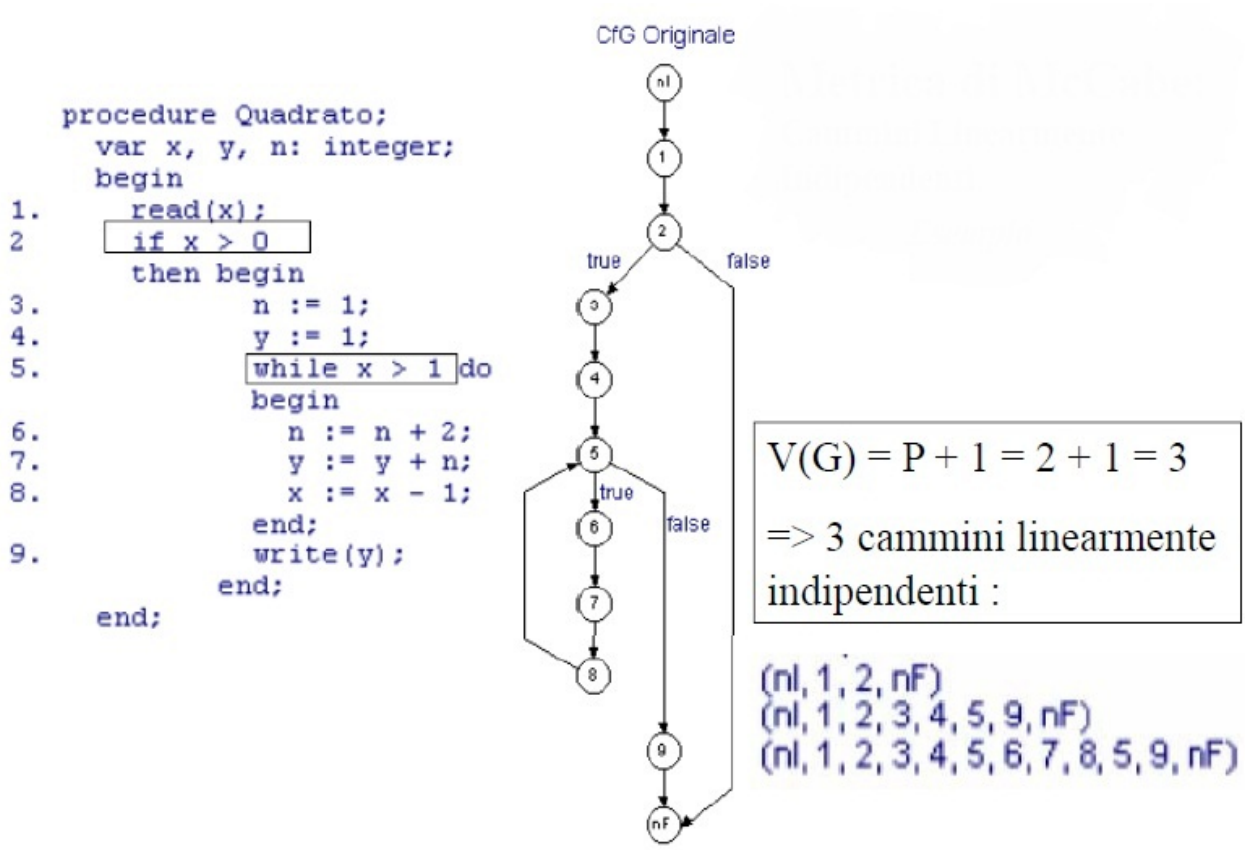


Abbiamo detto che il numero ciclomatico rappresenta il numero di percorsi linearmente indipendenti del Control Flow Graph, in particolare, secondo la teoria dei grafi, valgono le seguenti definizioni:

◆ Un cammino si dice indipendente se introduce almeno un nuovo insieme di istruzioni o una nuova condizione, rispetto ad un altro.

- In un CfG un cammino è indipendente se attraversa almeno un arco non ancora percorso dagli altri già considerati
- L'insieme di tutti i percorsi linearmente indipendenti di un'applicazione formano i *percorsi di base*
- Tutti gli altri possibili percorsi nel CfG sono generati da una combinazione lineare di quelli di base
- Dato un determinato programma l'insieme dei percorsi di base non è unico

I cammini linearmente indipendenti del nostro esempio saranno i seguenti:



Anche la metrica Cyclomatic Complexity, come nel caso della Lines of Code, presenta comunque diverse limitazioni e insufficienze, a ulteriore dimostrazione della inadeguatezza e della scarsa precisione dei risultati forniti dalle metriche maggiormente conosciute e utilizzate all'epoca dell'avvento della programmazione ad oggetti.

Come era sempre più crescente l'apprezzamento della filosofia di approccio, tipico della programmazione ad oggetti, così lo era anche la necessità di poter disporre di nuovi strumenti più adeguati e consoni, per ottenere misure più precise ed adeguate alla complessità, alle nuove funzionalità ed alle potenzialità tipiche delle nuove categorie di prodotti software.
Nel 1991 venne introdotta, da **Chidamber e Kemerer**, una nuova suite di metriche object-oriented che presero il nome di **Metriche CK**, dai loro ideatori.
Le metriche CK sono sei e, ad oggi, presentano ancora molteplici definizioni, a prova della continua evoluzione che la programmazione object-oriented sta tuttora registrando.
I professionisti, infatti, non hanno raggiunto un totale accordo su definzioni e algoritmi di calcolo comuni.
Le principali metriche CK sono le seguenti:

- **Weighted Method per Class (WMC)**
- **Depth of Inheritance Tree (DIT)**
- **Number Of Children (NOC)**
- **Coupling Between Object Class (CBO)**
- **Response For a Class (RFC)**
- **Lack Of Cohesion in Methods (LCOM)**

Una classe definisce N metodi di complessità c1, c2, c3, …, cn e la metrica Weighted Method per Class, di conseguenza, può essere definita come la somma delle complessità di tutti i metodi delle classi (calcolate tramite la Cyclomatic Complexity), più precisamente:

$$WMC = \sum c_i \quad i = 1,..,n$$

Un'ulteriore definizione vede la WMC come il numero dei metodi implementati in una classe, considerando in questo caso, un grado di

complessità unitario e sfruttando le complessità ciclomatiche dei singoli metodi, in particolare la formula è la seguente:

$$WMC(c) = \sum_{m \in M_{Im}(c)} \mathrm{VG}(m)$$

con **c** ad indicare il grado di complessità prescelto e **VG(m)** il numero ciclomatico del metodo **m**.

La prima misura è difficilmente implementabile, a causa dell'ereditarietà, infatti non tutti i metodi sono accessibili all'interno della gerarchia di classe.

Il numero dei metodi e la loro complessità sono un chiaro indice dello sforzo e del tempo necessario per curare la manutenzione di una classe.

Maggiore è il numero dei metodi all'interno di una classe e maggiore sarà l'impatto potenziale sui "figli", in quanto questi andranno ad ereditare tutti i metodi definiti dalla classe "padre", magari con l'aggiunta di ulteriore complessità.

Le classi con una grande numero di metodi risultano più specifiche per una determinata applicazione ed incidono fortemente sulla possibilità di riuso delle stesse.

La metrica **Depth of Inheritance Tree** o **Profondità dell'albero di ereditarietà** indica la distanza massima di un nodo (o classe) dalla radice dell'albero rappresentante la struttura ereditaria.

La si misura in classi "padre" e, all'aumentare della profondità della classe all'interno della struttura gerarchica, aumenta il numero di metodi che essa può ereditare, rendendo, di conseguenza, sempre più complessa una predizione precisa del suo comportamento.

Più gli alberi di ereditarietà presenteranno una profondità accentuata, più aumenterà la complessità del progetto, in quanto verranno chiamati più metodi e più classi, con l'incremento delle possibilità di riuso dei metodi ereditati.

Finalità principale di questa metrica è la misurazione dell'efficienza e della possibilità di riutilizzo senza che venga tralasciata la semplicità di comprensione e la manutenibilità.

Una metrica di supporto alla DIT è la **Number Of Children**, essa rappresenta il numero di sottoclassi immediatamente subordinate ad un determinata classe appartenente all'albero di ereditarietà.
In base alla variazione del valore ottenuto dalla NOC, si può determinare il livello di riutilizzabilità che garantisce una particolare classe al progetto e al sistema di dominio in generale.
La metrica NOC, a seconda del valore fornito, comporta la probabilità di avere un'astrazione errata delle classi "padre" indicando, a volte erroneamente, un impiego improprio delle relative sottoclassi.
Inoltre, al crescere del NOC, cresce, di conseguenza, la quantità di test-case necessari per testare correttamente le classi "figlie", il che comporta un incremento non trascurabile del tempo necessario per il testing.
La metrica NOC permette di avere, quindi, una valutazione dell'efficienza, della riutilizzabilità e della manutenibilità del prodotto software.
La metrica **Coupling Between Object Classes**, riassume il numero di collaborazioni o accoppiamenti che accomuna una classe con le altre appartenenti alla struttura interna dell'applicazione.
Un eccessivo accoppiamento tra classi può risultare dannoso per la modularità e la riutilizzabilità, infatti, al crescere del grado d'indipendenza di una classe corrisponde una maggior facilità di reimpiego in future nuove implementazioni.
Al crescere degli accoppiamenti, invece, si ha l'aumento dell'instabilità e della sensibilità alle modifiche delle parti componenti, il che comporta difficoltà crescenti nella manutenzione e nell'estendibilità del progetto.
Un sistema che presenta un alto numero di accoppiamenti raggiunge un livello di complessità crescente, causa principale delle molteplici difficoltà nella comprensione, nelle modifiche e nelle correzioni che si verificano nelle singole classi coinvolte negli accoppiamenti.
Focalizzando l'attenzione sulla problematica relativa all'eccessiva complessità del progetto in esame, si può intervenire limitando al minimo il numero di accoppiamenti tra classi allo scopo di migliorarne la modularità e di favorirne l'incapsulamento.

La metrica **Response For a Class** o **Risposta per classe**, è definita come la cardinalità (numero di elementi) dell'insieme dei metodi che possono essere richiamati in risposta ad un messaggio ricevuto da un determinato oggetto di classe o mandato da un metodo della classe stessa.
Questo include tutti i metodi accessibili all'interno della gerarchia della classe.
La metrica RFC, basandosi sul numero dei metodi, fornisce importanti informazioni riguardo alla complessità progettuale della classe ed alle possibilità di interazione e di comunicazione con le altre classi del sistema.
Maggiore è il numero di metodi che la classe ha la possibilità di richiamare e maggiore sarà la complessità della classe stessa.
A seconda del numero di metodi chiamati, in risposta ad un determinato messaggio, aumenta lo sforzo richiesto per il testing ed il debugging della classe e questo richiede ulteriori capacità di comprensione ed attenzione da parte dello sviluppatore.
La metrica RFC, infine, fornisce un'indicazione sulla compensibilità, sulla manutenibilità e sulla collaudabilità del progetto in esame.
L'ultima importante metrica caratteristica appartenente alla suite di Chidamber e Kemerer prende il nome di **Lack Of Cohesion Methods** o **Carenza di coesione dei metodi**.
La metrica LCOM indica il grado di coesione e diversità tra i metodi di una classe in riferimento alle proprietà strutturali (variabili e attributi) utilizzati dagli stessi.
Il risultato della LCOM permette di individuare possibili difetti di progettazione delle classi.
Esistono almeno due modi per calcolare la coesione tramite la LCOM:
il primo metodo proposto consiste, partendo prima di tutto dal presupposto che, per ogni campo dati di una classe si possa calcolare la percentuale di metodi che sfruttano tale campo dati, nell'effettuazione di una media delle percentuali così ottenute e alle quali viene, infine, sottratto il 100%.
Un percentuale che risulti bassa sarà indice di una alta coesione di dati e metodi tra classi.

Infine si procede contando gli insiemi disgiunti prodotti dall'intersezione fra insiemi di attributi usati dal metodo.

Più i metodi operano sugli stessi attributi e più essi risultano simili.

Sia **C** una classe con **N** metodi M1, M2, M3, …, Mn e sia $\mathbf{I_j}$ l'insieme delle variabili usate dal metodo $\mathbf{M_i}$.

Definiamo i seguenti insiemi:

$$P = \{ (I_i , I_j) \mid I_i \cap I_j = \emptyset \} \text{ e } \qquad Q = \{ (I_i , I_j) \mid I_i \cap I_j \neq \emptyset \}$$

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

Ad esempio:

si consideri una classe C con 3 metodi M1, M2 , M3.

Sia: $\{I_1\} = \{a, b, c, d, e\}, \quad \{I_2\} = \{a, b, e\} \quad \text{e } \{I_3\} = \{x, y, z\}$

Si ha: $I_1 \cap I_2 \neq \emptyset \quad \text{ma} \quad I_1 \cap I_3 = \emptyset \quad \text{e} \quad I_2 \cap I_3 = \emptyset$

LCOM = (numero intersezioni vuote) – (numero intersezioni non vuote) = 1

In seguito verrà mostrata una seconda definizione della metrica LCOM proposta da **Henderson - Sellers**:

$$LCOM(c) = \frac{\left(\frac{1}{a}\sum_{f \in F_D(c)} \mu(f)\right) - n}{1 - n}$$

Dove:

- **a** = |**FD(c)**| corrisponde all'insieme dei Field (attributi) dichiarati (e non ereditati) nella classe **c**
- **n** = |**M ...(c)**| corrisponde al numero dei metodi implementati nella classe **c**
- **μ(g)** = |{**m ϵ M …(c) : g ϵ FR(m)**}| corrisponde al numero di metodi implementati nella classe **c** che referenziano l'attributo **g** e FR(m) consiste nell'insieme dei metodi della classe **c** che referenziano **g**.

Se il valore della LCOM risultasse uguale a 0 indicherebbe una perfetta coesione e quindi tutti gli attributi della classe in oggetto sarebbero acceduti da tutti i metodi della classe stessa, mentre, se risultasse uguale ad 1, indicherebbe una completa assenza di coesione, comunicando all'utilizzatore che ogni attributo della classe è acceduto da un solo metodo della classe stessa.

Durante il processo di sviluppo del progetto una coesione scarsa o addirittura assente comporterebbe un aumento consistente delle complessità e, di conseguenza, della probabilità di errori e complicazioni; di solito si preferisce rimediare suddividendo le classi con bassa coesione in due o più sottoclassi con una coesione maggiormente consistente.

Concentrarsi sulla coesione tra metodi all'interno delle classi, vuol dire promuovere l'incapsulamento e la metrica LCOM si dimostra un efficiente strumento di misura dell'efficienza e della riusabilità di un progetto object-oriented.

E' importante sottolineare che non tutte le metriche mostrate vantano un livello di utilizzo talmente esteso da rendere disponibili criteri di valutazione delle misurazioni che risultino sufficientemente sperimentati e affidabili.

E' per tale motivo che si decide di utilizzare solo un sottoinsieme di tali metriche per misurare la qualità, l'accettabilità e l'affidabilità del software prodotto.

Per questo motivo, per ogni metrica, è stata definito un **valore di accettabilità** che indica la soglia di accettabilità dell'oggetto o dell'insieme degli oggetti misurati (nel caso si tratti di una media).

Per alcune metriche, inoltre, è stato definito un **valore di attenzione** che, pur non comportando la non accettabilità dell'oggetto o dell'insieme degli oggetti misurati, sollecita l'esecuzione di attività di indagine mirate ad individuare l'esistenza di problematiche e/o di potenziali rischi.

Di seguito è riportata una tabella riassuntiva:

Metrica	Descrizione	Valore di accettabilità	Valore di attenzione
AV(g)	Media della Complessità Ciclomatica dei metodi della classe	≤6	5,5≤AV(g) ≤6
LOC	Lines of Code	≤30	28≤LOC ≤30
LCOM	Lines of COMment	≥30%	30≤LCOM ≤32
WMC	Weighted Methods Per Class	≤14	12≤WMC ≤14
RFC	Response For a Class	≤100	90≤RFC ≤100
LOCM	Lack of Cohesion of Methods	≥75	75≤LOCM ≤80
CBO	Coupling Between Object classes	≤2	=2
DEPTH	Depth	≤7	6≤DEPTH ≤7
NOC	Number of Children	≤3	=3

CAPITOLO 5

Le metriche Object-Oriented: rappresentazione e interpretazione grafica delle metriche

Una volta che si è ponderato attentamente su quale metrica fare affidamento per le proprie esigenze progettuali, poter disporre anche di ingenti quantità di informazioni e risultati non basta per intervenire accuratamente ed in modo risolutivo sulle problematiche e sulle limitazioni progettuali emerse.

I valori, i dati e i risultati derivanti dalle misurazioni delle metriche devono essere prima correttamente catalogati, riconosciuti ed interpretati per poter rappresentare un vero e proprio ausilio alla progettazione.

Per questo motivo la NASA, nel 1992, creò il **Software Assurance Technology Center (SATC)**, come parte del **Systems Reliability and Safety Office** presso il **NASA Goddard Space Flight Center (GSFC)**.

Il SATC è stato fondato con l'intento di diventare un centro di eccellenza nella sicurezza software e nell'introduzione di notevoli migliorie sia nella qualità che nell'affidabilità del software sviluppato per la NASA presso il GSFC.

Il contributo del SATC, ci ha permesso innanzitutto di poter disporre di ormai fondamentali linee direttive di interpretazione basate sul confronto dei valori delle metriche fra i moduli usati come test e i moduli in esame.

In generale, individuare delle differenze tra i valori, anche di una certa entità, non significa necessariamente che i moduli in esame non siano efficienti, ma magari, che sia semplicemente necessaria un'analisi più approfondita.

Nella tabella seguente sono riportate quelli che il SATC ha reputato, in base agli studi di ricerca ed ai test effettuati, i valori ottimali per alcune delle metriche illustrate precedentemente:

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)

Come si può notare, nel caso delle metriche DIT e NOC, esiste una relazione di inversa proporzionalità (trade-off) fra metriche differenti: in questo particolare caso, infatti, un valore elevato di DIT può comportare una maggiore complessità in diversi fattori come, ad esempio, la manutenibilità, ma, al tempo stesso, consente di ottenere una migliore riutilizzabilità del prodotto software.

Le metriche, come sottolineato più volte, sono sempre da considerare supporti di misurazione, specializzati nel fornire indici qualitativi riguardo alle differenti proprietà caratterizzanti il progetto in esame.

Starà sempre, comunque, al programmatore la responsabilità finale di tenere a mente le relazioni tra le strutture e l'effetto che comporta la variazione dei valori espressi da una particolare metrica; infatti, le scelte effettuate durante tutto il processo di sviluppo comporteranno ripercussioni su molti aspetti fondamentali quali testing, qualità, affidabilità, comprensibilità, manutenibilità e riutilizzabilità.

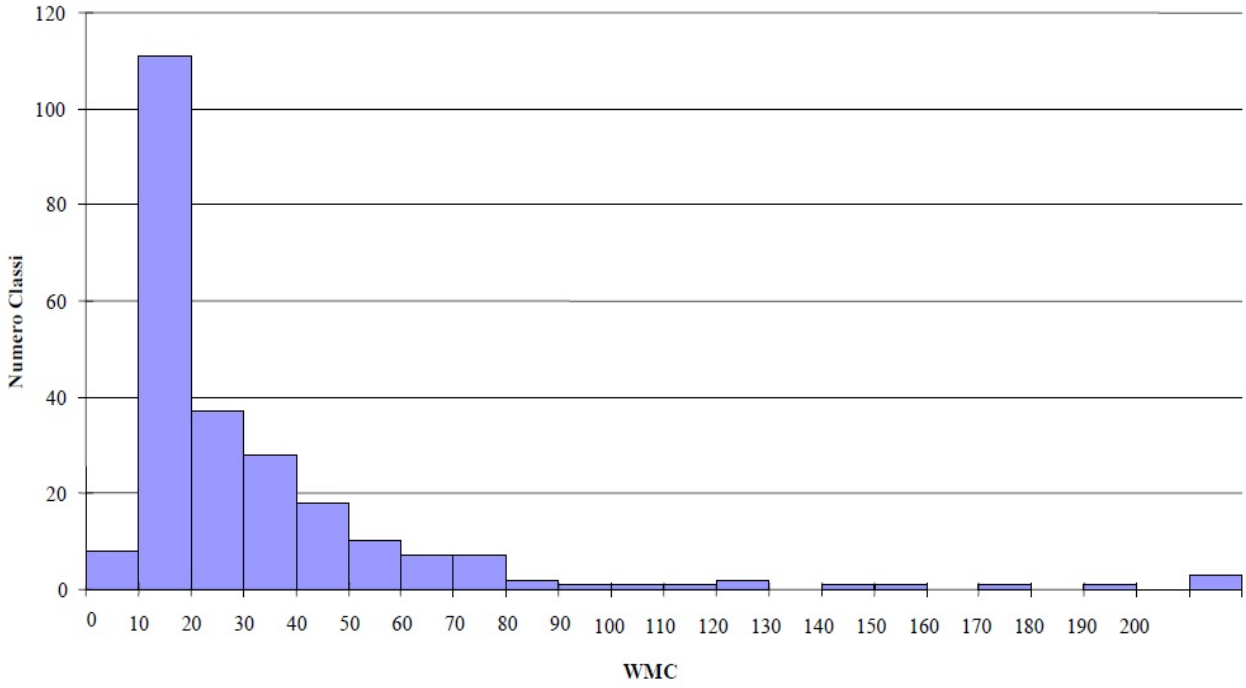
Grazie alle ricerce effettuate presso i centri di ricerca del SATC, è possibile rappresentare ed interpretare chiaramente e con criterio i valori assunti dalle metriche software, soprattutto dalle metriche CK.

Nell'esempio in figura si può notare un istogramma che rappresenta, per ogni valore di WMC, quante classi del progetto hanno tale valore.

In questo caso si può notare che, mentre molte classi riportano un valore WMC<20, alcune riportano un valore di WMC>100.

Nel caso di queste ultime, quindi, sarà necessario ulteriori controlli e correzioni.

L'istogramma è utile anche ai fini del monitoraggio della complessità totale.

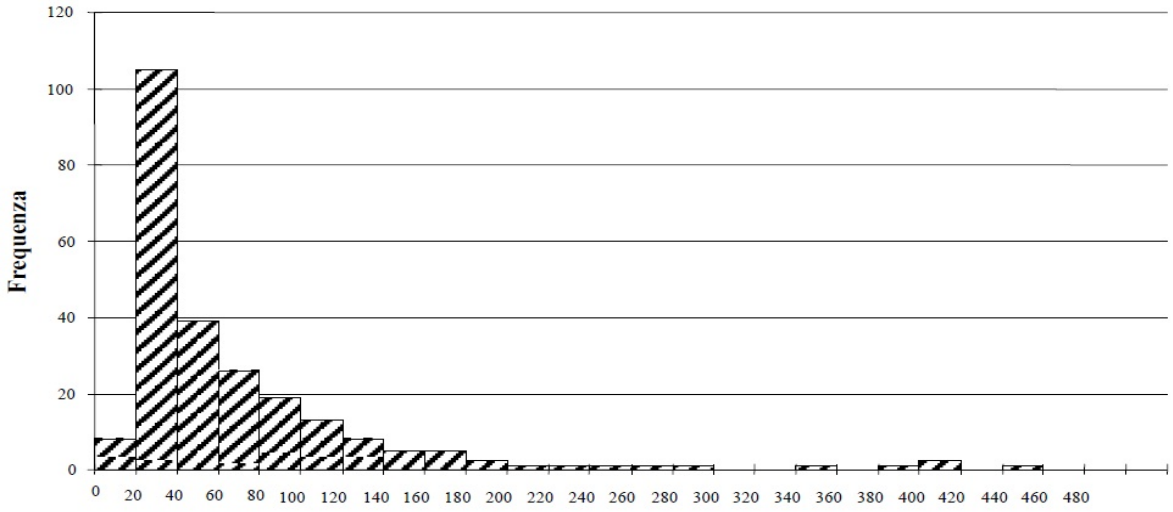


Come nel caso della metrica WMC, ovviamente, anche per le altre metriche, è possibile tracciare istogrammi simili.

L'istogramma sottostante, per esempio, è riferito alla metrica RFC e rappresenta un progetto in cui alcune classi sono capaci di richiamare più di 200 metodi.

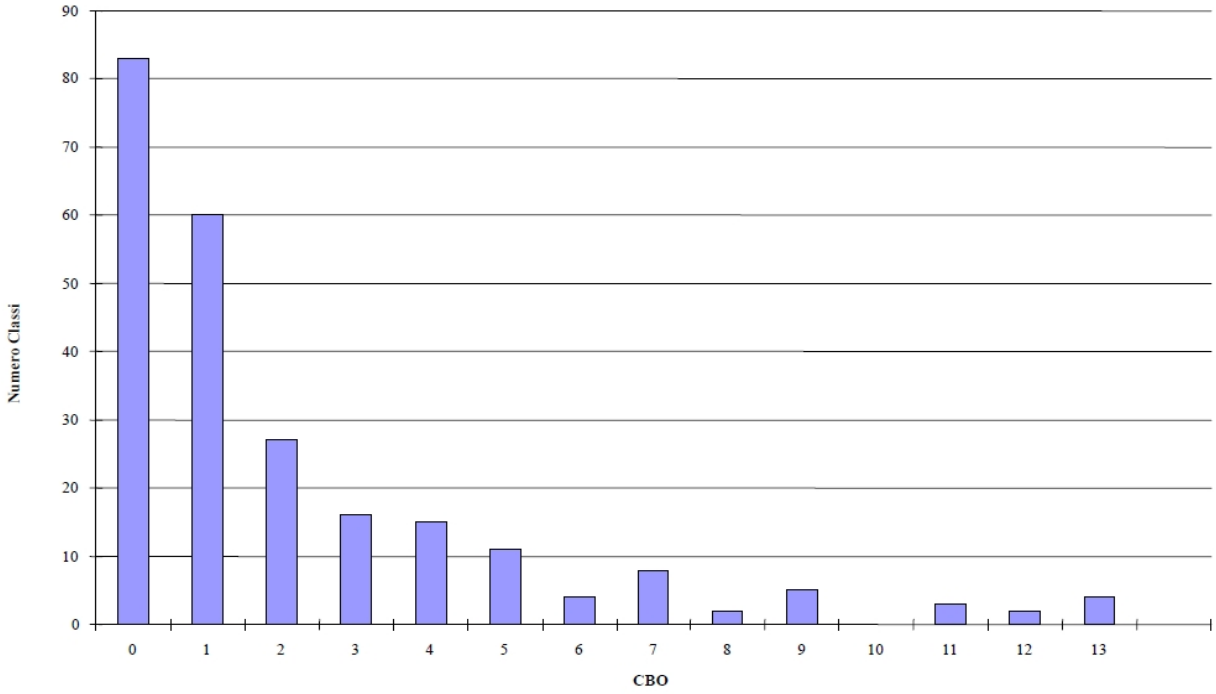
Classi per presentano valori molto elevati di RFC dimostrano un'elevata complessità sfavorendo quindi comprensibilità, testing e debugging.

Anche l'istogramma seguente è di utilità nel monitoraggio della complessità.



Di seguito è riportato un istogramma relativo ad una misurazione effettuata con la metrica CBO: il progetto conta 240 classi e più di un terzo sono indipendenti in quanto hanno un valore di CBO pari a 0.

Alti valori di CBO sono indice di scarsa comprensibilità, riutilizzabilità e manutenibilità.

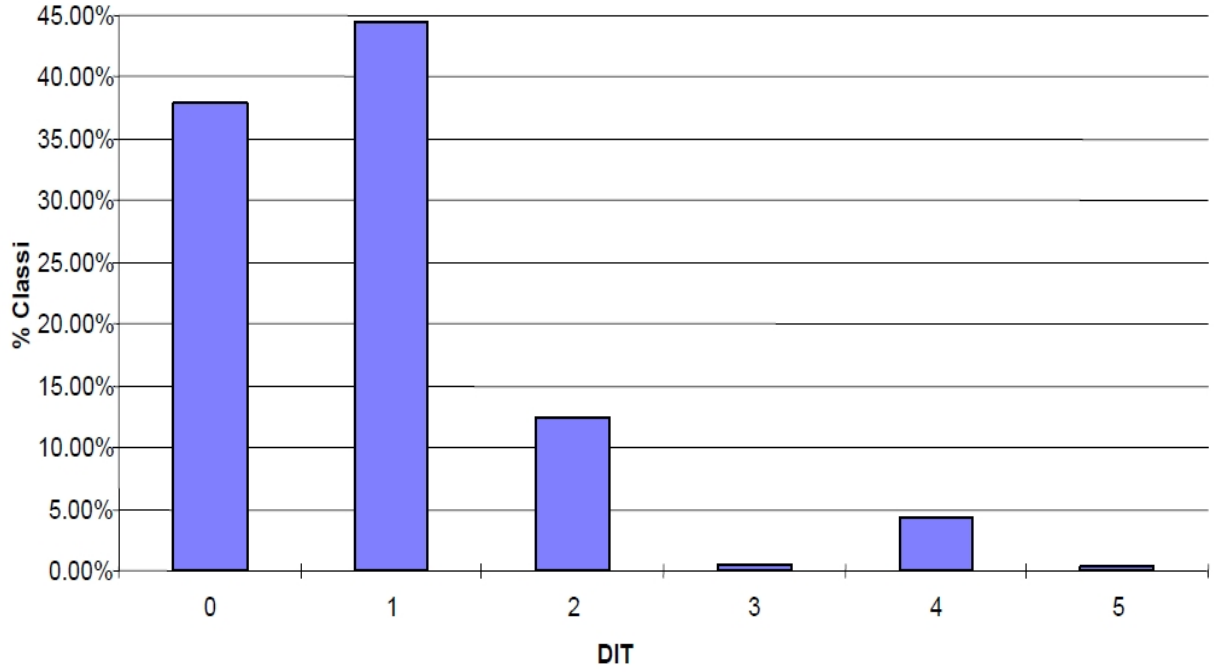


A livello grafico possiamo notare di seguito le modalità di rappresentazione grafica di metriche riferite alla struttura gerarchica come le metriche NOC e DIT.

Una classe con valore DIT pari a 0 rappresenta la radice dell'albero e, se essa fosse anche una foglia, cioè riportasse un valore NOC pari a 0, si tratterebbe di un unico codice che non porterebbe alcun beneficio in favore di ereditarietà e riutilizzabilità.

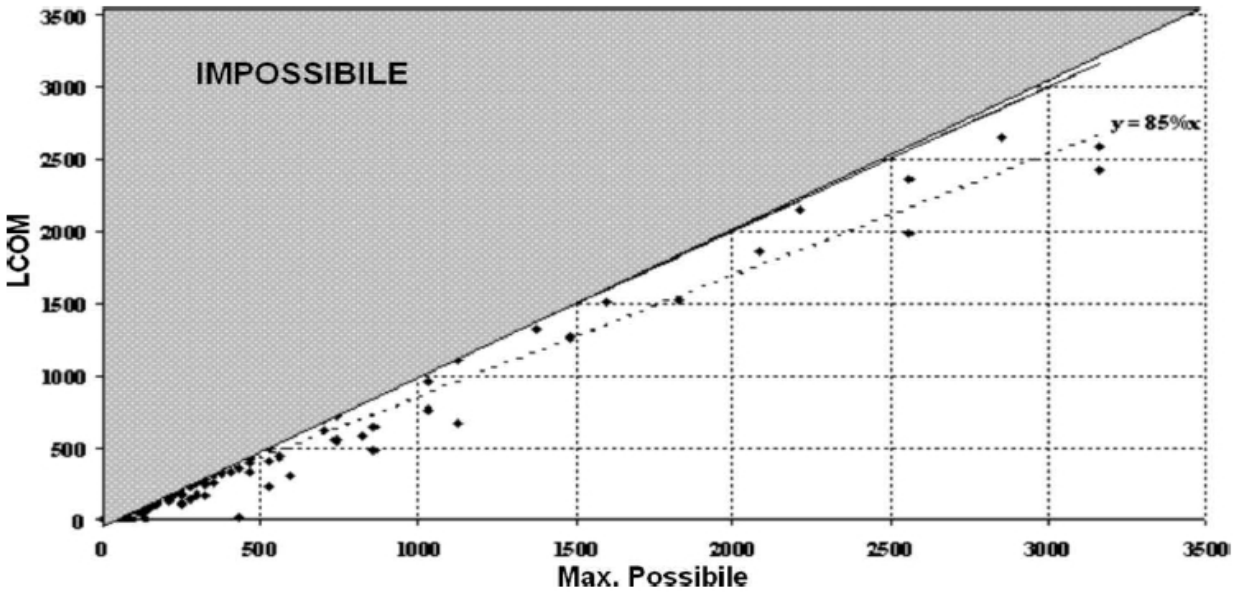
Nell'esempio mostrato, quasi il 66% delle classi si trovano al di sotto di altre classi dell'albero ed indica un livello moderato di riutilizzabilità.

Percentuali più alte di 2/3% per la metrica DIT rappresenterebbero un elevato grado di riutilizzabilità ed anche una crescente complessità.



Come introdotto nelle pagine precedenti, il valore della metrica LCOM è dipendente dal numero di metodi, perciò dovrà esistere un valore massimo possibile.

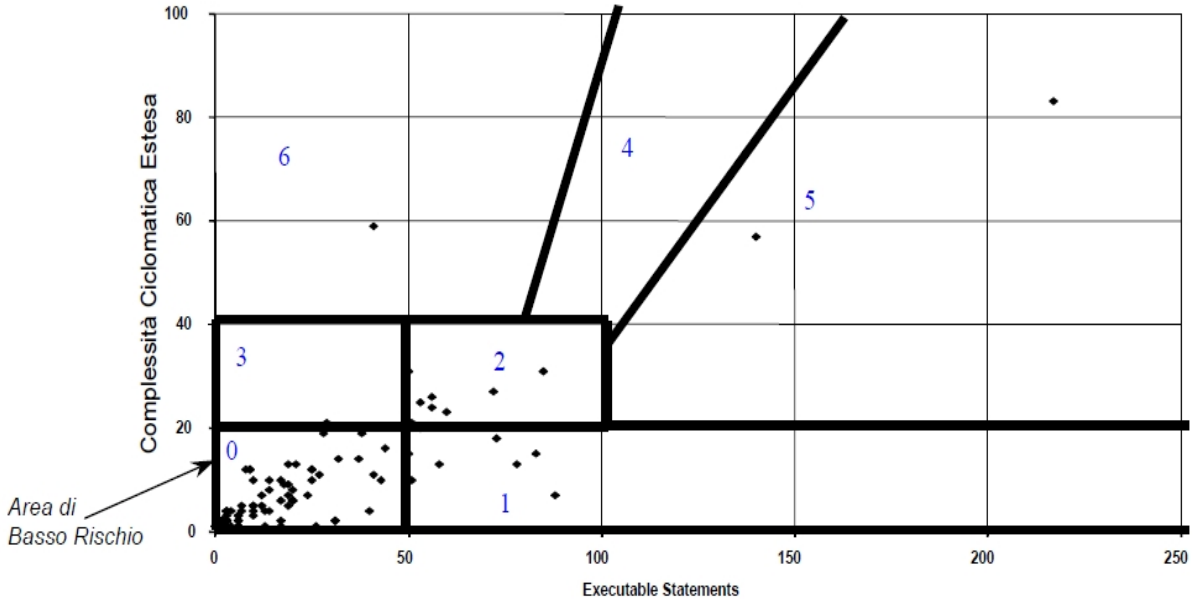
Nel grafico seguente vengono riportati i valori LCOM rispetto al loro massimo possibile.



Il valore ideale di LCOM per un determinato progetto, confrontato con il suo valore massimo, sarà il più piccolo possibile.

Molto spesso, per un'analisi più accurata e approfondita, si procede alla misurazione incrociando i risultati di più metriche per valutare la qualità dei moduli in esame.

Il SATC ha eseguito diversi studi di ricerca per identificare i valori di riferimento e le cosiddette "regioni a rischio" che rappresentano le zone in cui i metodi mostrano scarsa qualità e vanno ad influire su manutenibilità, riutilizzabilità e compensibilità.



I risultati ottenuti:

	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

CAPITOLO 6

Le metriche Object-Oriented: i tool software principali

Avere la completa conoscenza delle capacità e funzionalità delle singole metriche impiegate e sapere correttamente interpretare e catalogare i risultati raccolti non sempre può bastare per ottenere un'analisi precisa e completa del codice esaminato.

Ad oggi, poter disporre di strumenti che, non solo permettano di applicare nel modo più corretto, rapido ed efficiente possibile le metriche software più importanti, ma anche poter analizzare i contenuti derivanti da esse fornendo all'utilizzatore risultati completi, chiari e facilmente interpretabili ed impiegabili, rappresenta un notevole guadagno in termini di tempo, di costi e di efficienza dei prodotti software realizzati.

I tool software per l'impiego delle metriche software sul codice generato hanno fatto la loro comparsa pochissimo tempo dopo l'inizio dell'adozione in massa da parte dei progettisti software, sempre più interessati ai benefici e ai nuovi standard di qualità che avrebbero potuto garantire questi nuovi metri di misurazione.

Ovviamente uno dei primi linguaggi impiegati nella realizzazione dei primissimi tool software, soprattutto nel contesto della programmazione object-oriented, è stato il linguaggio C++, che ha rapidamente sostituito nell'impiego il suo predecessore, il linguaggio C.

Ponendo particolare attenzione soprattutto ai risultati forniti e ad una corretta applicazione delle metriche impiegate, i primi tool creati, verso la fine degli anni '90, erano di tipologia a console, perciò graficamente spartani e privi di capacità di output tabellare e grafico particolarmente avanzati, ma le metriche impiegate erano già molte tra quelle più utilizzate ed apprezzate come: la Lines of Code, la Cyclomatic Complexity e le CK Metrics.

Il primo tool che merita di essere menzionato prende il nome di **C and C++ Code Counter** o **CCCC.**

L'inizio dei lavori per lo sviluppo di questo tool risale all'anno 1997 per opera di Tim Littlefaire ancora oggi, esso è apprezzato per la sua semplicità d'utilizzo ed efficienza, confermandosi come uno dei più impiegati nell'analisi di prodotti software realizzati nei linguaggi C, C++ e Java.

CCCC è inoltre un metric tool completo che permette l'impiego di metriche fondamentali quali: LOC, Cyclomatic Complexity e le CK Metrics.

Successivamente sono nati altri due tool ancora oggi molto famosi ed apprezzati: **Unified CodeCount** o **UCC** e **COCOMO II**, il primo è un toolset che implementa gli standard di "code counting" stabiliti dal Software Engineering Institute (SEI) e che si basa fondamentalmente sull'impego della metrica Source Lines of Code (SLOC), permettendo di stabilire con chiarezza caratteristiche come la capacità di differenziazione e livello della complessità globale dei sorgenti del codice.

UCC consente l'impiego in diversi linguaggi di programmazione come: Ada, ASP, ASP.NET, C#, C/C++, ColdFusion, CSS, HTML, Java, JavaScript, JSP, Perl, PhP, SQL, VB, VbScript, XML, Bash Script, ColdFusion, C Shell Script, Fortran, NeXtMidas, Python e Xmidas.

Il secondo software menzionato invece, arrivato alla sua seconda versione, è un pratico tool metrico per la stima di costi, tempi e forza lavoro richiesti per l'intero prodotto analizzato.

I metrics tool più recenti dimostrano una sempre maggiore possibilità di interazione con i risultati metrici ottenuti e garantiscono anche una forte interoperabilità con gli ambienti di sviluppo maggiormente utilizzati.

Non possiamo esimerci dal menzionare tool, come **Source Monitor**, **Nitriq** ed **NDepend**, strumenti completi, graficamente avanzati rispetto ai loro predecessori a console e maggiormente evoluti negli algoritmi di calcolo.

Source Monitor e un tool freeware, scritto in C++, che consente di ottenere rapidamente risultati metrici attraverso i file sorgenti, e che supporta alcuni tra i maggiori linguaggi come: C++, C, C#, VB.NET, Java, Delphi, Visual Basic (VB6) or HTML e include metriche anche a livello di metodi e funzioni.

Carattestica interessante offerta da questo tool è quella di poter conservare e confrontare le diverse analisi eseguite nel corso di sviluppo di un determinato prodotto software, potendone ricavare i risultati ottenuti tramite le correzioni suggerite o, per esempio, il subentro di possibili nuove problematiche dovute ad errori, complessità aggiuntive o difficoltoso raggiungimento degli obiettivi qualitativi preposti.

SourceMonitor consente di stampare e salvare tabelle, grafici e gli utilissimi diagrammi di Kiviat, un ausilio grafico particolare che racchiude al suo interno tutti i principali valori, con le rispettive soglie di riferimento, da garantire alla conclusione dello sviluppo software.

Non è da meno la possibilità di poter disporre di una GUI apposita e del supporto all'utilizzo di script tramite l'uso di comandi XML.

Nitriq è un metric tool freeware, di recente uscita, appartenente alla categoria dei **LinqToCode tools**, cioè, da come suggerisce il termine, a quei tool che sfruttano una serie di query in linguaggio Linq ottimizzate appositamente per applicare metriche come LOC o Cyclomatic Complexity nel modo più chiaro ed efficiente possibile in particolare agli assembly C#.

Oltre alle metriche principali, Nitriq offre una vasta gamma di query proprietarie pensate appositamente nella rilevazione di problematiche legate al design, alla correttezza e alla complessità nel codice analizzato.

Oltre ad offrire utili informazioni riassuntive sulle caratteristiche e statistiche del software in analisi Nitriq consente di avere a disposizione delle “treemap” grafiche accurate e complete, consentendo agli sviluppatori di poter immediatamente concentrarsi sui componenti più problematici a livello di LOC e Cyclomatic Complexity.

Il massimo livello di integrazione con l'ambiente di sviluppo Microsoft Visual Studio, completezza, facilità d'utilizzo ed applicazione nell'ambito dei metric tools è rappresentato da NDepend.

NDepend è lo stato dell'arte raggiunto in questi decenni di evoluzione delle metriche software e di tutti i maggiori tool software d'applicazione.

NDepend supporta nativamente il linguaggio C# e tutte le potenzialità offerte dal Framework .NET, ma esistono anche altre due versioni appositamente pensate per il supporto ai linguaggi Java e C++, JArchitect e CppDepend.

Affidandosi a questo software si può avere accesso, tramite un vastissima gamma di query Linq messe a disposizione dell'utilizzatore, a grafi e matrici delle dipendenze ed a 82 metriche differenti tra cui: Lines of Code, Cyclomatic Complexity, Depth Of Inheritance Tree, Number Of Children e Lack Of Cohesion.

La forza di questo tool metrico sta soprattutto nella possibilità, per la prima volta, di poter interagire con il codice in esame partendo direttamente dai risultati forniti dalle differenti metriche applicate, fornendo agli sviluppatori, in unico ecosistema, la potenza e l'utilità delle metriche software unite alle funzionalità ed agli strumenti che solo un vero ambiente di sviluppo può garantire.

Gli output di NDepend sono sia esportabili in differenti tipologie di formati sia integrabili con la documentazione relativa al codice prodotto ed esaminato e questo consente maggiore completezza e chiarezza, ma anche maggiore manutenibilità ed estendibilità garantite.

CAPITOLO 7

Le metriche Object-Oriented: esempio di test ed applicazioni pratiche

Dopo aver saggiato le funzionalità e le potenzialità delle metriche software, i benefici dovuti alla corretta applicazione dei risultati derivanti da esse e la praticità e l'utilità che rappresentano i tool software di supporto esistenti, non resta che presentare un'analisi reale a dimostrazione della validità delle argomentazioni trattate.

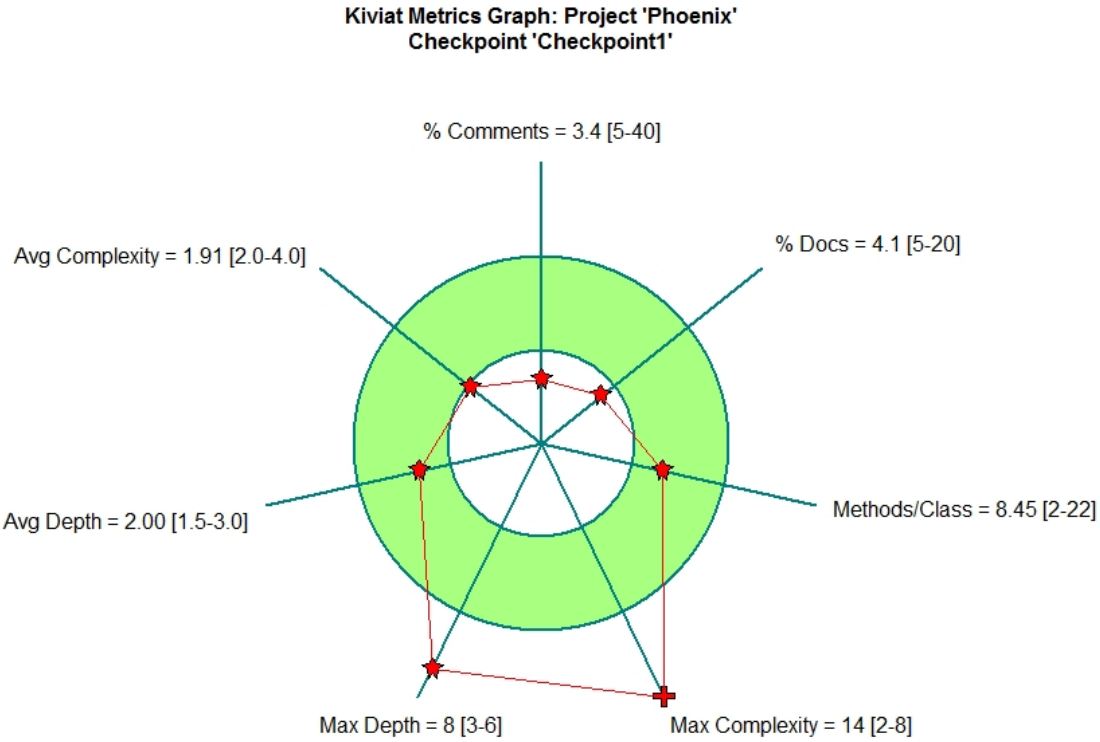
Il prodotto software, oggetto dell'analisi, consiste in un framework il cui codice è scritto in linguaggio C# e tuttora trova diverse applicazioni in ambito accademico.

Il prodotto software è stato esaminato, dal punto di vista metrico, con i tre tool ad interfaccia grafica presentati nelle pagine precedenti: SourceMonitor, Nitriq e NDpend.

Per ogni tool software impiegato verranno messi in rilievo i relativi output enfatizzando le informazioni utili derivanti da essi.

I primi risultati raccolti nel processo di analisi metrico sono stati quelli derivanti dall'impiego del software SourceMonitor: particolarmente interessante risulta il diagramma di Kiviat che sottolinea le seguenti problematiche:

1) La percentuale di commenti nel codice è inferiore alla soglia consigliata in relazione alla dimensione del codice in analisi;
2) Il codice registra la presenza di componenti che superano i valori di soglia di Cyclomatic Complexity consigliata;
3) La profondità massima di alcuni componenti nel codice supera i valori di soglia di Depth Of Inheritance Tree consigliata.

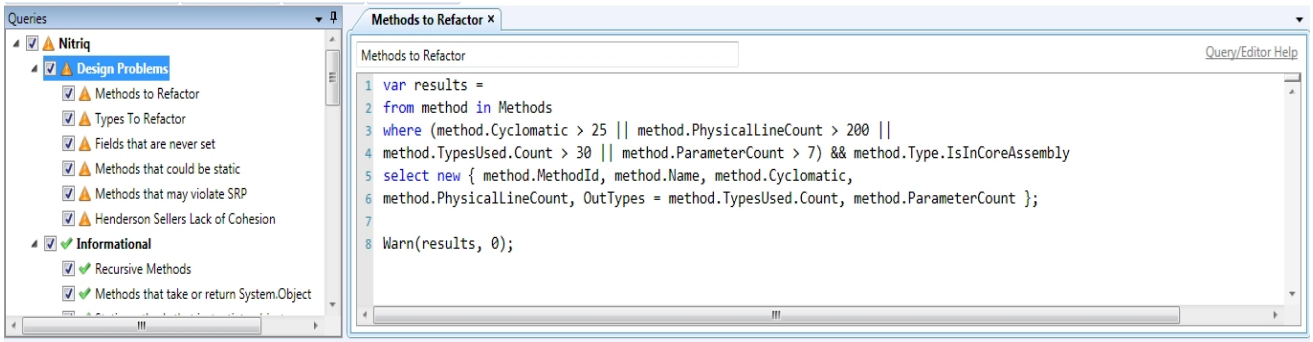


Procedendo con l'analisi tramite il metric tool Nitriq, tramite la sua interfaccia grafica, notiamo tre differenti finestre d'interesse:

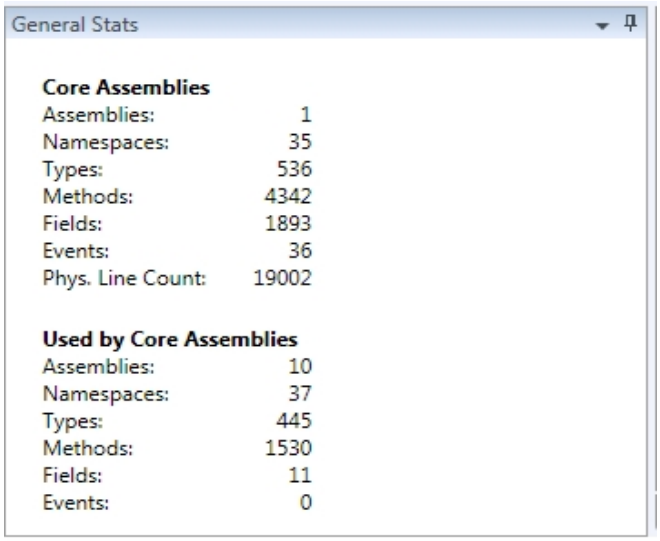
La prima, la “Queries Window”, riassume tutta la gamma di possibili query Linq effettuabili sul codice da analizzare e nel riquadro accanto è possibile vedere il dettaglio di ogni singola query.

La suite di query di test è divisa in due categorie principali a loro volta suddivise in più sottocategorie:

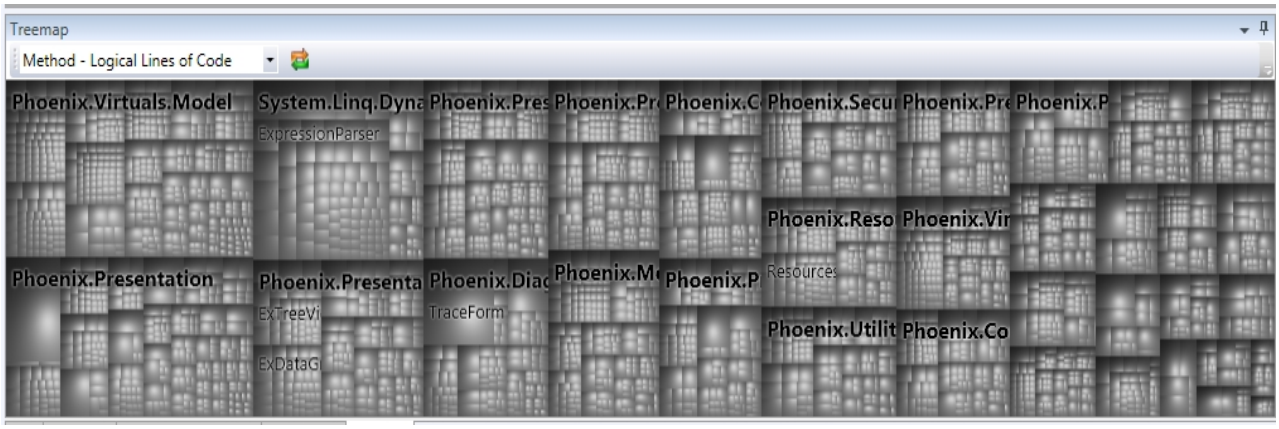
- **Nitriq**, organizzata in: **Design Problems** e **Informational**
- **NX Cop**, organizzata in: **Design**, **Maintainability** e **Naming**



La seconda, la "General Stats Window", presenta allo sviluppatore alcune informazioni basilari sul codice in esame come, ad esempio, il numero di metodi, tipi e campi e il numero di linee di codice fisiche.



La terza e ultima finestra, la "Treemap Window", da come sugerisce il nome, consente allo sviluppatore di poter evidenziare, tramite l'apposita rappresentazione a riquadri di tutto l'assembly, i componenti maggiormente problematici secondo moltissimi indici metrici di riferimento e di poter, così, successivamente intervenire su determinati punti del codice prodotto tramite l'ambiente di sviluppo.



Per completare l'analisi e poter disporre del maggior numero di dati e informazioni utili possibili, riguardanti soprattutto gli aspetti qualitativi del codice prodotto, ho voluto mettere alla prova le potenzialità del tool software NDepend, uno dei più completi ed avanzati metric tool attualmente in commercio.

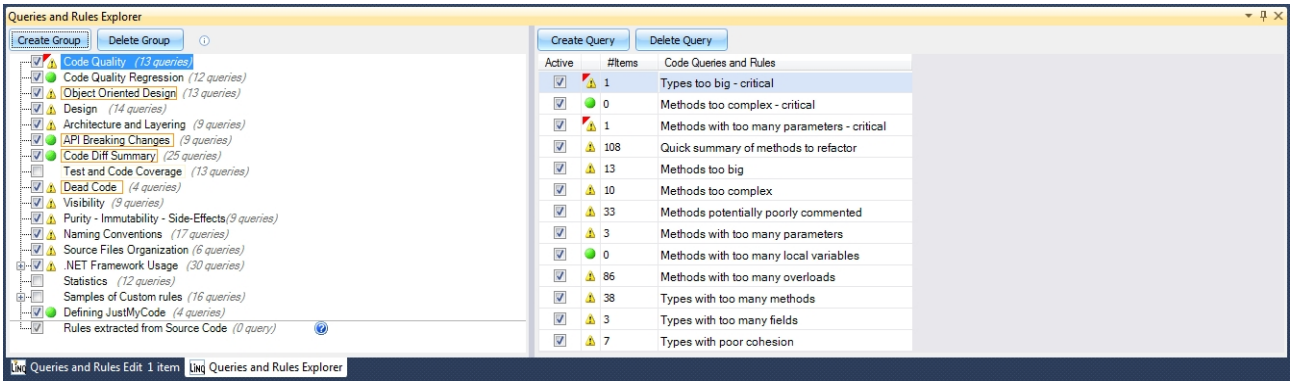
Una delle maggiori caratteristiche di NDepend è l'ampia integrabilità garantita: infatti, lavorare con NDepend equivale a dire lavorare direttamente con il proprio ambiente di sviluppo, unificando sia la analisi qualitativa delle metriche sia tecnica del codice minimizzando al massimo i tempi di risoluzione e modifica delle problematiche riscontrate legate alla qualità, alla manutenibilità e alla complessità ed evidenziate nel processo d'analisi.

NDepend racchiude una suite completa di algoritmi metrici di analisi che offrono risultati, grafici e proposte di soluzioni davvero chiari, efficienti e facilmente applicabili.

Il primo componente alla base del funzionamento del tool è il “Query and Rules Explorer”, che offre, oltre alla possibilità di creare le metric query personalizzate con l'apposito Editor, una vasta gamma di informazioni relative allo stato di salute del software analizzato nelle seguenti categorie di riferimento:

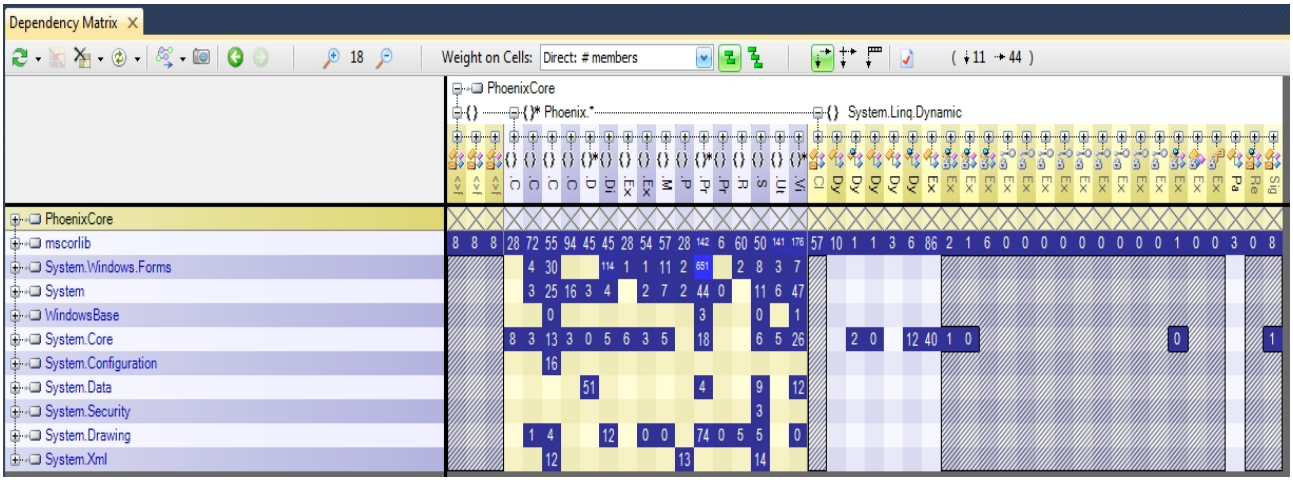
- **Code Quality**;
- **Object Oriented Design**;
- **Test and Code Coverage**;
- **Visibility**;

- **Source Files Organization;**
- **.NET Framework Usage**;

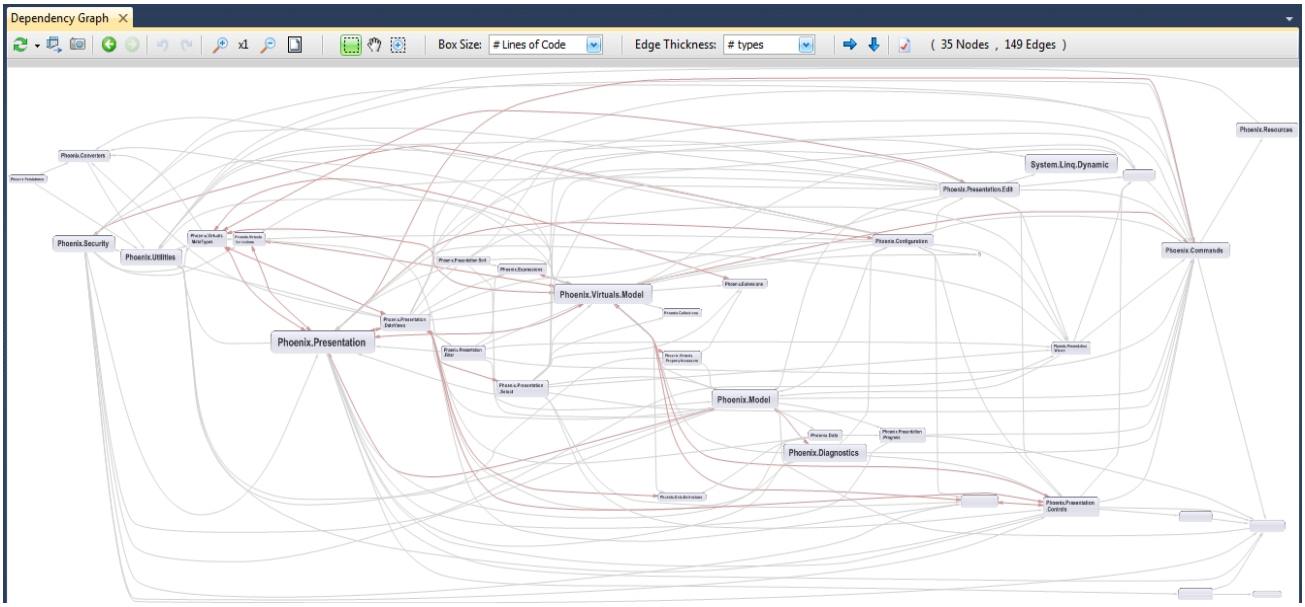


Dal punto di vista grafico offre tre eccellenti metodi di rappresentazione del codice in analisi:

1) La **Dependency Matrix**, una matrice delle dipendenze interne ed esterne all'interno del prodotto software

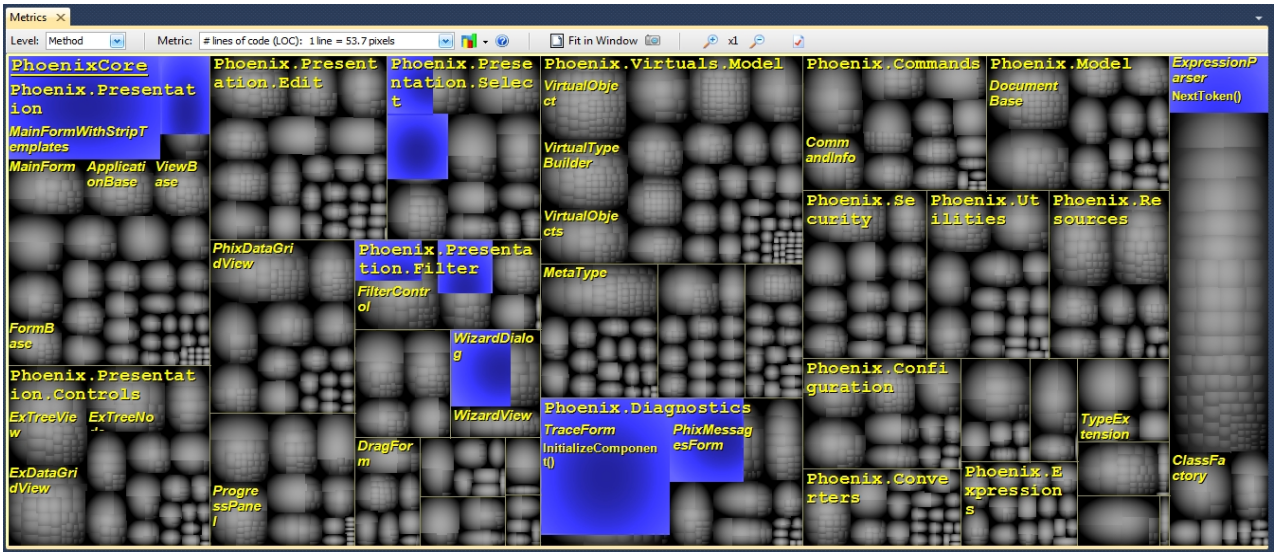


2) Il **Dependency Graph**, un grafico le cui linee rappresentano ogni singola dipendenza all'interno del codice rappresentando, in pratica, i risultati riportati in modo tabellare nella Dependency Matrix

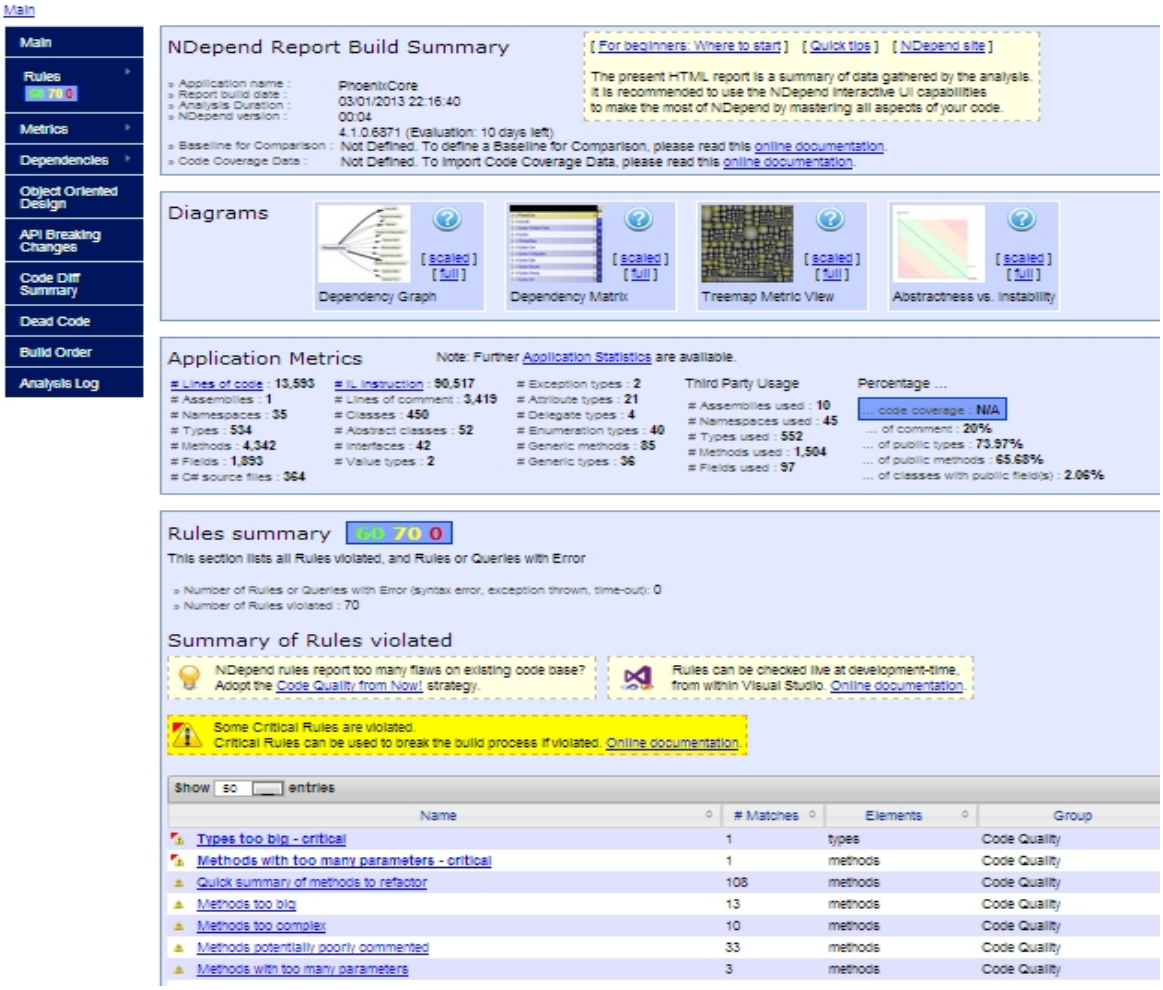


3) La **Metrics Window**, una finestra che mostra tutto il codice, sottoforma di riquadri e sottoriquadri riprendendo la gerarchia che caratterizza il prodotto software.

Attraverso questa particolare modalità di visualizzazione non solo è possibile ricavare risultati derivanti da analisi metriche come la LOC, la Cyclomatic Complexity o la Nesting Depth, ma anche ricavare, in tempo reale, una catalogazione precisa delle classi, dei metodi, dei campi o dei tipi per indici, in modo tale da poter individuare immediatamente i casi che meritano particolare attenzione e cautela e poter, così, intervenire tramite i suggerimenti mirati offerti dal tool NDepend.



Pregio di NDepend è anche quello di garantire la conservazione dei dati e la portabilità dei contenuti a prescindere dall'ambiente e dal contesto di sviluppo in quanto, quale resoconto delle analisi effettuate, essa permette la redazione automatica di pratiche e complete documentazioni Html che consentono la fruizione e la presentazione dei contenuti in un formato universale e l iberamente consultabile.



BIBLIOGRAFIA

1) **L. Westfall. "12 Steps to Useful Software Metrics"** http://floors-outlet.com/specs/spec-t-1-20111117171200.pdf.

2) **I. Caballero, and E. Verbo. "A Data Quality Measurement Information Model Based on ISO/IEC 15939"** http://mitiq.mit.edu/iciq/pdf/a%20data%20quality%20measurement%20information%20model%20based%20on%20iso--iec%2015939.pdf.

3) **G. H. Travassos. "A Quantitative Approach to Software Management and Engineering"** http://www.cs.umd.edu/~mvz/mswe607-f00/notes-5b1.pdf

4) **C. Jones. "A Short History of Lines of Code (LOC) Metrics"** https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=2&ved=0CD8QFjAB&url=http%3A%2F%2Fwww.result-planning.com%2Fdl187&ei=JVg3UdvlM8TIsgafy4H4Cg&usg=AFQjCNE7GYupNsg0yfqUaRB-Ml2bLI9aJQ&sig2=lAC8PTeT3bHXKZFJPC6oAg&bvm=bv.43287494,d.Yms.

5) **D. Rodriguez, and R. Harrison. "An Overview of Object-Oriented Design Metrics"** http://www.cc.uah.es/drg/b/RodHarRama00.English.pdf.

6) **Y. Zhou, and H. Leung. "Analysis of CK Metrics"** https://docs.google.com/viewer?url=http%3A%2F%2Fwww.compapp.dcu.ie%2F~renaat%2Fca421%2FAnalysis%2520of%2520CK%2520Metrics.ppt.

7) **C. Cruz, and A. Erika. "Chidamber & Kemerer Suite of Metrics"** https://docs.google.com/viewer?url=http%3A%2F%2Fsdlab.naist.jp%2Fmembers%2Fcamargo%2Fpresentations%2FCKMetrics.ppt.

8) **M. Hitz, and B. Montazeri. "Chidamber & Kemerer's Metrics Suite: A Measurement Theory Perspective"** https://www.google.it/url?sa=t&rct=j&q=&esrc=s&source=web&cd=1&cad=rja&ved=0CD0QFjAA&url=http%3A%2F%2Fciteseerx.ist.psu.edu%2Fviewdoc%2Fdownload%3Fdoi%3D10.1.1.52.7366%26rep%3Drep1%26type%3Dpdf&ei=BFs3Ua7XNszItAbm-IHwBg&usg=AFQjCNEuHoKbf2LriIABtqOPR6xdG61Mkg&sig2=ZXi0y5o8hpanHmAYgQjOhg&bvm=bv.43287494,d.Yms.

9) **J. J. Al-Ja'afer, and K. E. M. Sabri. "Chidamber-Kemerer (CK) and Lorenze-Kidd (LK) Metrics to Assess Java Programs"** http://www.cas.mcmaster.ca/~sabrike/wp-content/uploads/2008/02/iwss2004.pdf.

10) **R. Lincke, J. Lundberg, and W. Löwe. "Comparing Software Metrics Tools"** http://www.cs.umd.edu/~pugh/ISSTA08/issta2008/p131.pdf.

11) **A. Kaur, S. Singh, K. S. Kahlon, and P. S. Sandhu. "Empirical Analysis of CK & MOOD Metric Suit"** http://ijimt.org/papers/78-M451.pdf.

12) **U. Bellur. "Evaluating Design Goodness – Metrics"** https://docs.google.com/viewer?url=http%3A%2F%2Fwww.it.iitb.ac.in%2F~palwencha%2Fse%2Fse%2Fslides%2FSession5-EvaluatingDesigns-OOMetrics.ppt.

13) **E. Tramontana. "Evoluzione Delle Metriche"** http://www.dmi.unict.it/~tramonta/se/L19_EvoluzMetriche.pdf.

14) **A. Saporito. "Ingegneria Del Software: Le Metriche Software"** http://wwwusers.di.uniroma1.it/~ingsoft1/Ingegneria_del_Software_1.pdf.

15) **W. Meding, and M. Staron. "ISO/IEC 15939 Creating an Efficient Measurement Program"**

16) **G.Raiss. "La Valutazione Del Software. Le Norme ISO/IEC 14598 e 15939"** http://www2.cnipa.gov.it/site/_contentfiles/01379900/1379952_ISO%2014598.pdf.

17) **R. Sbavaglia. "Le Metriche Ed Il Loro Utilizzo Nello Sviluppo Del Software"** http://torlone.dia.uniroma3.it/sistelab/annipassati/sbavaglia.pdf.

18) **E. Colonese. "Metriche." In *Manuale Di Sviluppo Software***

19) **E. Lamma. "Metriche Del Software"** http://www-lia.deis.unibo.it/Courses/IngSW/SE_C_2_Metriche_.pdf.

20) **G. Armano. "Metriche Del Software"** http://www.jugsardegna.org/vqwiki/jsp/Wiki?action=action_view_attachment&attachment=MetricheDelSoftware-F029.pdf.

21) **G. A. Di Lucca. "Metriche Per Il Modello Di Analisi"** http://www.ing.unisannio.it/dilucca/ISW/materiale1112/MetricheCVS.pdf.

22) **G. A. Di Lucca. "Metriche Per Sistemi Object-Oriented"** http://www.ing.unisannio.it/dilucca/LSISW/materiale0809/MetricheOO.pdf.

23) **G. A. Di Lucca. "Misurare Il Software"** http://www.ing.unisannio.it/dilucca/GSSW/materiale09/metriche.pdf.

24) **V. Ambriola, G. A. Cignomi, C. Montangero, and L. Semini. "Misurazione Del Software"** http://www.math.unipd.it/~tullio/IS-1/2004/Dispense/P19.pdf.

25) **M. Lorenz, and J. Kidd. *Object-Oriented Software Metrics***

26) **R. B. Grady. *Practical Software Metrics for Project Management and Process Improvement***

27) **A. Colombo, E.Damiani, and F. Frati. "Processo Di Sviluppo Software e Metriche Correlate: Metamodello Dei Dati e Architettura Di Analisi"** http://www.crema.unimi.it/Biblioteca/Note_pdf/127.pdf.

28) **M. Scotto, Alberto Sillitti, Giancarlo Succi, and Tullio Vernazza. "Raccolta Non Invasiva Di Metriche Di Podotto"** http://www.inf.unibz.it/~gsucci/publications/images/scotto-sillitti-succi-vernazza_camera_ready.pdf.

29) **C. Kaner, and W. Bond. "Software Engineering Metrics: What Do They Measure and How Do We Know?"** http://testingeducation.org/a/metrics2004.pdf.

30) **A.Cimitile. "Software Maesurement"** http://www.ing.unisannio.it/cimitile/ing-soft/dispense/metriche.PDF.

31) **A.Finkelstein. "Software Metrics"** http://www0.cs.ucl.ac.uk/staff/A.Finkelstein/advmsc/11.pdf.

32) **N. Leveson. "Software Metrics"** http://sunnyday.mit.edu/16.355/classnotes-metrics.pdf.