****

# ISW – Gruppo 01 – TeamSoftwareRevolution

Metrica – Linee di codice

Redattori: Carta Stefano 48554;

Carta Fabio 65006;

Argiolas Alessandro 49062;

Bertulu Giovanni 49044;

Desogus Omar 48975;

Fadda Luca 49006;

Zucca Luigi 65053;

# Sommario

1. Introduzione
2. Standard ISO/IEC 9126
   1. Modello delle qualità - ISO/IEC 9126-1
   2. Metriche delle qualità Esterne - ISO/IEC 9126-2
   3. Metriche delle qualità Interne - ISO/IEC 9126-3
   4. Metriche delle qualità in Uso - ISO/IEC 9126-4
3. Linee di Codice
   1. Analisi
   2. Esempio 1
   3. Esempio 2
   4. Esempio 3
4. Conclusioni

1.Introduzione

Lo scopo del presente elaborato è il trattamento di un aspetto fondamentale dell’ingegneria del software, il concetto di misura della qualità di un prodotto software.

Attraverso un costruttivo lavoro di gruppo, si è analizzato e discusso lo Standard ISO/IEC 9126 e si è cercato di capire come viene suddiviso.

Il team ha cercato una valida risposta al perché il numero di linee di codice non costituisce una misura della qualità di un prodotto software.

2.Standard ISO/IEC 9126

Apriamo questo capitolo ponendoci subito una domanda: Cos’è una metrica?

Ebbene **l’ISO** (*International Organization for Standardization)* definisce uno standard che individua le linee guida per descrivere un modello di valutazione per le qualità di un prodotto software.

Lo standard consta di quattro parti:

* Modello delle qualità
* Metriche di Qualità Esterne
* Metriche di Qualità Interne
* Metriche di Qualità in Uso

Il modello quindi, definisce le caratteristiche di qualità, con annesse sotto caratteristiche, dette anche attributi.

Il punto di vista delle diverse qualità cambia durante il ciclo di vita del SW, ad esempio, all’inizio del ciclo produttivo infatti, la qualità specificata dai requisiti descrive il concetto di qualità esterna al prodotto secondo il punto di vista degli utenti.

## 2.1 Modello delle qualità – ISO/IEC 9126-1

Prima di procedere con l’analisi dello standard è importante introdurre i concetti di *requisiti funzionale* e *non funzionale* o di qualità, questi ultimi sono materia fondamentale dell’ Ingegneria del software.

I **requisiti funzionali** descrivono le funzionalità del sistema software, in termini di servizi che il sistema software deve fornire, di come il sistema software reagisce a specifici tipi di input e di come si comporta in situazioni particolari.

I **requisiti non funzionali** descrivono le proprietà del sistema software in relazione a determinati servizi o funzioni e possono anche essere relativi al processo.

Tornando allo standard ISO/IEC 9126, il modello definisce sei caratteristiche principali con relative sotto-caratteristiche per definire le qualità di un software:

* Funzionalità
* Affidabilità
* Efficienza
* Usabilità
* Manutenibilità
* Portabilità

La funzionalità è l’unico requisito funzionale, gli altri cinque fanno parte dei requisiti non funzionali.

Vediamo più nel dettaglio le varie parti analizzandone anche le rispettive sotto-categorie.

## [Funzionalità](https://it.wikipedia.org/wiki/Funzionalit%C3%A0)

Capacità di fornire funzioni che soddisfano esigenze stabilite.

Di queste fanno parte:

* **Appropriatezza**: Capacità di fornire specifiche funzioni per soddisfare i compiti prefissati
* **Interoperabilità**: Capacità di operare con più sistemi differenti
* **Accuratezza**: Capacità di fornire i risultati aspettati
* **Sicurezza**: Capacità di proteggere informazioni e dati negando l’accesso ai non autorizzati, siano loro utenti o sistemi
* **Conformità**: Adesione a standard, convenzioni e regolamentazioni

## [Affidabilità](https://it.wikipedia.org/wiki/Funzionalit%C3%A0)

Capacità di mantenere determinate prestazioni per un certo lasso di tempo e in certe condizioni specifiche.

Di queste fanno parte:

* **Maturità**: Evitare che si verifichino errori o risultati non desiderati
* **Recuperabilità**: Capacità di ripristino dopo un malfunzionamento o un risultato inaspettato
* **Aderenze**: Aderenza a regole e standard riguardanti l’affidabilità
* **Tolleranza agli errori**: Mantenere un livello di prestazioni prefissato anche in presenta di malfunzionamenti o uso scorretto del sistema

## Efficenza

Capacità di fornire appropriate prestazioni relativamente alla quantità di risorse usate in condizioni prestabilite.

Di queste fanno parte:

* **Comportamento rispetto al tempo**: Fornire brevi tempi di risposta ed elaborazione in determinate condizioni
* **Utilizzo delle risorse**: Utilizzo di quantità e tipo di risorse adeguato
* **Conformità**: Adesione a standard sull’efficienza

## Usabilità

Capacità di essere appreso, capito e benaccetto dall’utente.

Di queste fanno parte:

* **Comprensibilità**: Facilità nel capire i concetti del SW e mettere l’utente in condizione di servirsene
* **Apprendibilità**: Capacità di riduzione del tempo richiesto all’utente per imparare ad usare e prendere familiarità con il sistema
* **Operabilità**: Capacità di utilizzare il software e controllarne l’uso per i propri scopi
* **Attrattiva**: Capacità di essere piacevole per l’utente che ne fa uso
* **Conformità**: Adesione a standard sull’usabilità

## Manutenibilità

Capacità di essere modificato, corretto, migliorato o adattato.

Di queste fanno parte:

* **Analizzabilità**: Facilità con un quale è possibile analizzare il codice alla ricerca di errori
* **Modificabilità**: Permettere l’implementazione di una specifica modifica
* **Stabilità**: Evitare risultati e comportamenti inaspettati dopo una modifica
* **Testabilità**: Facilità di testing per la validazione di modifiche apportate

## Portabilità

Capacità di essere trasportato e operare in ambienti diversi.

Di queste fanno parte:

* **Adattabilità**: Capacità di essere adattato per differenti ambienti operativi senza applicare modifiche
* **Installabilità**: Capacità del software di essere installato in uno specifico ambiente
* **Sostituibilità**: Capacità di essere sostituito con un prodotto che svolge gli stessi compiti
* **Conformità**: Adesione a standard sull’usabilità

Tutte queste caratteristiche e sotto-caratteristiche sono comprese all’interno delle metriche di qualità esterne e metriche di qualità interna.

## 2.1 Metriche di Qualità Esterne – ISO/IEC 9126-2

Le metriche di qualità esterne misurano i comportamenti del software sulla base dei test, dall'operatività e dall'osservazione durante la sua esecuzione, in funzione degli obiettivi stabiliti in un contesto tecnico.

## 2.1 Metriche di Qualità Interne – ISO/IEC 9126-3

Le metriche di qualità interne vengono applicate al software non eseguibile (ad esempio il [***codice sorgente***](https://it.wikipedia.org/wiki/Codice_sorgente)) durante le fasi di progettazione e codifica.

Permettono infatti l’individuazione di eventuali problemi che potrebbero influire sulla qualità finale del prodotto prima che sia realizzato il software eseguibile.

## 2.1 Metriche di Qualità in Uso – ISO/IEC 9126-4

Le metriche in uso rappresentano il punto di vista dell'utente sul software.

Il livello di Qualità in uso è raggiunto quando si è raggiunto sia il livello di qualità esterna sia il livello di qualità interna.

Di queste fanno parte:

* **Produttività**: mettere in grado gli utenti di spendere una quantità di risorse appropriate in relazione all'efficacia ottenuta
* **Efficacia**: Mettere in grado gli utenti di raggiungere gli obiettivi con accuratezza e completezza
* **Soddisfazione**: capacità del prodotto di soddisfare gli utenti
* **Sicurezza:** Raggiungere accettabili livelli di rischio di danni a persone, software, apparecchiature o all'ambiente d'uso

3.Linee di Codice

Analizzato lo standard ISO/IEC 9126 e le sue sotto categorie, possiamo interrogarci su quali metriche da noi trovate possono essere corrette o meno.

Alcune caratteristiche dello standard si riferiscono a ciò che l’utente vede, percepisce e capisce.

Altre, entrano nello specifico, analizzando e seguendo nei dettagli le varie fasi della vita del software fin dall’inizio della produzione.

Vi sono qualità che si occupano della manipolazione, modifica, miglioramento e adattamento del codice.

Parlando di codice quindi ci chiediamo, la quantità e quindi il numero delle linee di codice, è considerabile una misura della qualità di un prodotto software?

## 3.1 Analisi del problema

Ricapitolando:

Le qualità in uso descrivono e giudicano il software dal punto di vista dell’utente, che quindi non ha accesso al codice.

Di conseguenza sappiamo che non hanno a che fare con la struttura interna del prodotto.

Di seguito vengono identificate tra le sotto caratteristiche delle qualità esterne e delle qualità interne, quelle che si occupano di valutare il codice.

La **manutenibilità** è l’unica qualità che tratta la visione e la manipolazione del codice.

In particolare solo due sotto caratteristiche si riferiscono e agiscono direttamente sul codice:

**Modificabilità:** Implementazione di una specifica modifica (Sul codice)

**Analizzabilità:** Facilità con un quale analizzare il codice alla ricerca di errori

Nessuna di queste però parla della quantità di righe di codice.

Le linee di codice sono molto accurate perché misurano esattamente ciò che è stato editato, ma poco significative perché non dicono nulla sulla difficoltà reale del problema.

La lunghezza del codice e quindi il relativo numero di righe può dipendere da fattori quali l’ esperienza, la bravura e le scelte stilistiche .

Non tutti i programmatori possiedono le stesse conoscenze e competenze infatti, quello che può essere scritto in poche righe di codice da un programmatore esperto, sarà scritto da un programmatore inesperto con l’uso di un numero di righe maggiore.

In oltre, I programmatori possono scegliere quale stile adottare durante la stesura del codice.

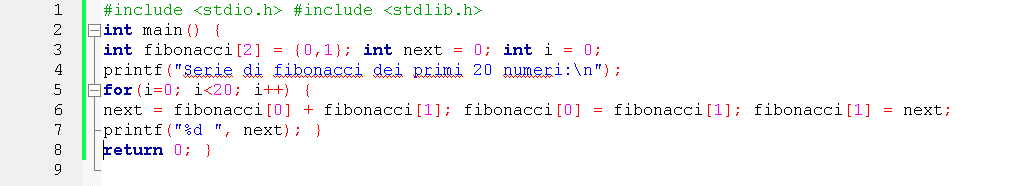
Un codice leggibile, il più delle volte, risulta più lungo mentre un codice più compatto possiamo adottare sintassi specifiche a discapito della comprensibilità.

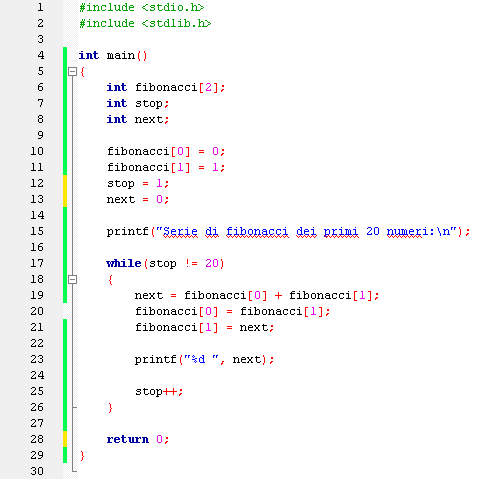
Seguono esempi di diversi codici implementati con diversi linguaggi di programmazione nelle quali si adottano svariate scelte stilistiche e tecniche.

## 3.2 Linguaggio Imperativo Strutturato

In questo paragrafo, vediamo come le scelte stilistiche e implementative possono comportare un incremento nelle linee di codice.

Vengono mostrati due esempi di codice raffiguranti l’implementazione della serie di Fibonacci in linguaggio C.





## 3.3 Linguaggio Imperativo ad Oggetti

In questo paragrafo, viene mostrata l’implementazione delle classi Persona e Studente.

Nel primo esempio (fig.2) vediamo la classe Studente con l’implementazione di tutti i suoi metodi e nel secondo (fig.3) la classe Studente che eredita i suoi metodi dalla classe Persona.

I codici differiscono per il numero di righe.

Fig. 1 Classe Persona

Fig. 2 Classe Studente

Fig. 3 Classe Studente che estende Persona

Fig.1 Fig.2



Fig.3



## 3.4 Linguaggio Funzionale

In questo paragrafo vediamo due funzioni implementate nel linguaggio funzionale ocaml.

Una recursive(fig.1) e una tail-recursive(fig.2) e della prima vedremo anche un implementazione con scelte stilistiche diverse(fig.3).



fig.1 fig.2 fig.3

4.Conclusioni

In conclusione, possiamo affermare che il numero di linee di codice non è classificabile tra le metriche usate per giudicare un prodotto software.

Come detto, il numero delle linee di codice, pur essendo calcolabili con assoluta precisione, possono mutare per vari fattori che non influenzano il fine del codice e il risultato atteso.