****

# ISW – Gruppo 01 – TeamSoftwareRevolution

Metrica

Redattori: Stefano Carta (T.L.);

Luigi Zucca;

Omar Desogus;

Giovanni Bertulu;

Fabio Carta;

Alessandro Argiolas;

Luca Fadda.

# Sommario

1. Introduzione
2. Complessità dei prodotti software
   1. Linguaggi imperativi
   2. Linguaggi Object-Oriented
3. Metriche
   1. Tempi di risposta
   2. Modularità
   3. Velocità di completamento
4. Errori
   1. Definizione
   2. Classificazione
   3. Testing
5. Discipline e fondamenti scientifici
6. Conclusioni

# Introduzione

Lo scopo del presente elaborato è il trattamento di alcuni aspetti fondamentali dell’ingegneria del software, quali la complessità dei prodotti software, le metriche di valutazione e tipologie di errori. Inoltre si è cercata una disciplina, che come l’ISW, sta cercando di trovare un proprio fondamento scientifico.

Attraverso un costruttivo lavoro di gruppo, partendo dalle conoscenze e dal lavoro individuale, sono stati argomentati gli aspetti sopra riportati, evidenziando sia i punti di forza che difetti e mancanze.

# Complessità dei prodotti software

Un prodotto software è rappresentato dall'insieme di programmi, procedure di realizzazione, documentazione di sistema, dati di configurazione e manuale di riferimento.

Un prodotto software può essere di due distinte tipologie:

* Generico: destinato all’uso di massa (general purpose)
* Specifico: realizzato per le esigenze di uno specifico utente/azienda (specific purpose)

La realizzazione di un prodotto software è un processo lungo, essa richiede la collaborazione di diverse figure professionali specializzate (sviluppatori, team leader, analisti ecc) le quali, insieme agli investitori, agli utenti ed altre figure interessate ,rappresentano gli stakeholders del prodotto.

Quest'ultimo, nella generalità dei casi, risulta molto complesso, pertanto necessita di una suddivisione in moduli, ciascuno dei quali viene implementato e curato da un diverso team di sviluppo.

Un possibile criterio per la valutazione della complessità di un prodotto software è rappresentato dal numero di moduli che lo compongono, pertanto risulta evidente che la complessità cresce in maniera proporzionale all’aumentare dei moduli.

La dimensione di un programma, inteso come numero di righe di codice, pur avendo un impatto visivo notevole, non rappresenta sicuramente un indice di valutazione della complessità.

Due programmi che risolvono lo stesso problema, aventi la stessa quantità di dati in ingresso, differenziati dal numero di righe, possono fornire entrambi la stessa soluzione.

A conferma di questa tesi, esistono diversi stili di programmazione che si differenziano per la compattezza del codice.

Diversi programmatori preferiscono un codice più “esteso”, di conseguenza più leggibile, piuttosto che un codice più compatto e di difficile interpretazione.

## 2.1 Linguaggi imperativi

Le figure seguenti (fig. 2.1 e fig. 2.2) forniscono un esempio di due codici che risolvono lo stesso problema utilizzando scelte sintattiche e stilistiche differenti:

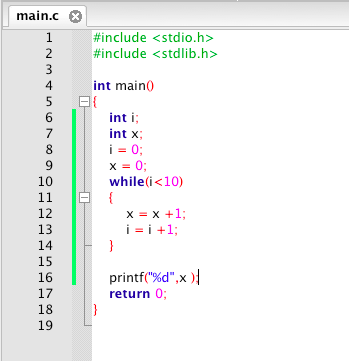
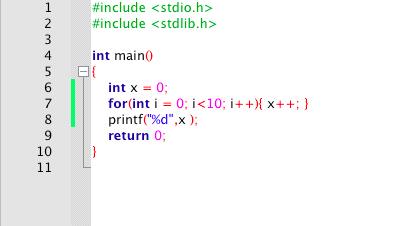


fig. 2.1 fig. 2.2

Come è possibile notare, la figura 2.1 mostra un codice avente circa la metà delle linee di codice presente nel programma di fig. 2.2.

## 2.2 Linguaggi Objcet-Oriented

Vediamo ora un esempio legato alla programmazione ad oggetti:

Nelle figure seguenti (fig.2.3 e fig.2.4) vengono rappresentate rispettivamente le classi animale e gatto con i relativi attributi e metodi.

Notiamo che le due classi possiedono gli stessi attributi e lo stesso metodo e sappiamo che un oggetto gatto è logicamente sottoclasse di animale.

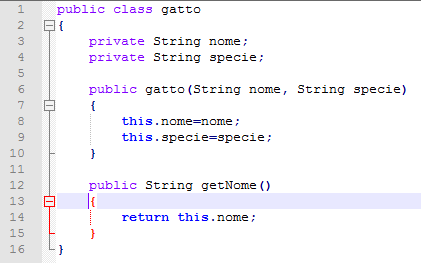
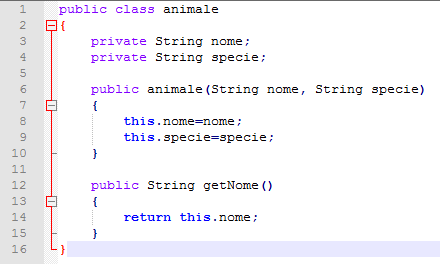


fig.2.3 fig.2.4

Osservando le figure precedenti è possibile notare come le due classi siano rappresentate mediante attributi e metodi identici.

Questa scelta implementativa può causare, nel caso di programmi lunghi e complessi, un enorme spreco di righe e tempo.

Nella figura seguente (fig.2.5), è mostrata una nuova implementazione della classe “gatto” che utilizza il principio di ereditarietà, tipico della programmazione ad oggetti.

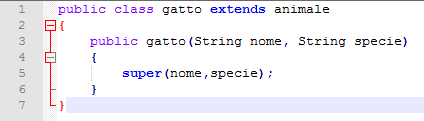


fig.2.5

# Metriche

Nel paragrafo 2 abbiamo introdotto il concetto di valutazione della complessità di un prodotto software, ma come possiamo valutarne la qualità?

Prima di elencare varie alternative è doveroso introdurre il concetto di **metrica.**

Una metrica è uno standard per la misurazione di alcune proprietà legate al prodotto software.

Queste metriche possono essere suddivise in due categorie:

* Metriche esterne
* Metriche interne

Le metriche esterne o di prodotto stabiliscono le caratteristiche principali per la produzione di un software:

* Costi
* Tempi
* Produttività

Le metriche interne o di progetto permettono la valutazione qualitativa del software:

* Funzionabilità
* Manutenibilità
* Usabilità

Introdotto il concetto di metrica, possiamo ora elencarne alcune.

## 3.1 Tempi di risposta

Una buona metrica per giudicare la qualità di un SW è la misurazione dei tempi di risposta, i quali dipendono dal tipo di algoritmo utilizzato (complessità computazionale) .

Tuttavia, questa metrica è influenzata dall'hardware dell'elaboratore su cui è installata l'applicazione, pertanto risulta impossibile valutare in termini assoluti la metrica in questione.

## 3.2 Modularità

La modularità, come già accennato nel paragrafo 2, consiste nel suddividere il problema complesso in sottoproblemi. Poiché un prodotto software ha grandi dimensioni, risulta evidente la necessità di suddividerlo in moduli, secondo il principio "divide et impera".

Il deficit della modularità è rappresentato dal fatto che, a volte, visto che i diversi moduli possono essere sviluppati da programmatori diversi, è possibile che si verifichino errori dovuti alla scarsa comunicazione tra i componenti del team.

## 3.3 Velocità di completamento

Per determinare se un prodotto è di buona qualità, possiamo anche valutare la velocità di completamento di un’azione, intesa come numero di task.

Questa metrica, presa singolarmente, risulta inefficace poiché non tutti gli utenti che utilizzano un dato prodotto software possiedono le stesse caratteristiche fisiche, logiche e informatiche.

# Errori

Dopo aver introdotto le metriche, abbiamo visto alcuni casi d'uso di queste ultime per la valutazione di un prodotto software e i difetti ad esse legate.

Ogni problema nel software rappresenta una mancanza di valore e un’ opportunità per migliorarlo. Classificare i problemi del software permette di **affrontarli** e **risolverli**.

Tra i tanti problemi che possiamo riscontrare durante il processo di “creazione” di un prodotto software ci sono gli errori.

# 4.1 Definizione

Un errore o bacco (in inglese bug) rappresenta un "difetto" che si ha nell'esecuzione di un programma o di un prodotto software, quando, per certi tipi di dati in ingresso, si ha un funzionamento diverso da quello voluto.

Gli errori di differenziano per casi e tipologie.

## 4.2 Classificazione

* Requisiti difettosi
  + Errati o incompleti
* Incomprensione cliente-sviluppatore
* Errori progettuali
  + Mancanza di specifiche, errori algoritmici
* Errori durante il testing
  + Testing affrettati, correzione parziale degli errori
* Errori di documentazione
  + Documentazione parziale o inesistente

**4.3 Testing**

La tecnica del testing consiste in un collaudo del prodotto software, ovvero nell'eseguire una serie di esperimenti al fine di minimizzare la probabilità (cercando di farla tendere a zero) di avere un malfunzionamento. La fase di testing è fondamentale per la ricerca di errori, pertanto potrebbe risultare lunga e onerosa. La durata di questa fase dipende dallo scopo del prodotto software. Se ad esempio il prodotto è rivolto all'intrattenimento ci si può accontentare di un livello di testing meno spinto rispetto ad uno in ambito aerospaziale.

## 5. Discipline e fondamenti scientifici

Come tante altre discipline, anche l'Ingegneria del Software è alla ricerca di un fondamento scientifico. Tali discipline, dette empiriche, applicano tecniche sviluppate dai professionisti piuttosto che dai teorici.

Una disciplina che è alla ricerca di un fondamento scientifico è la Psicoanalisi, di cui Sigmund Freud è considerato il padre. La Psicoanalisi, appartenente alle cosiddette Pseudoscienze, cerca di descrivere i comportamenti dell'uomo attraverso un approccio basato sull'osservazione diretta dei pazienti e non su teoremi.

**6. Conclusioni**

Nel presente elaborato, mediante un lavoro di gruppo sinergico, si sono affrontate tematiche inerenti concetti fondamentali dell’ ingegneria del software.

Partendo dalle nostre conoscenze di base e da informazioni apprese durante le ore di lezione, le quali hanno fornito un valido spunto di riflessione, abbiamo discusso e approfondito le suddette tematiche.

Nello specifico, abbiamo appreso il concetto di metrica, specificandone alcune e fornendone gli aspetti critici.

Oltre alle metriche abbiamo imparato a classificare le varie tipologie di errori individuando nella fase di testing un metodo per determinarne la quantità.

Infine, si è ricercata una disciplina scientifica che analogamente all’ ingegneria del software non è basata su fondamenti teorici.

Tra numerose discipline che continuano ad applicare tecniche sviluppate dai professionisti invece che dai teorici, troviamo la Psicoanalisi, la quale basa i risultati ottenuti sull’osservazione e non su teoremi.