

ALMA MATER STUDIORUM – UNIVERSITÁ DI BOLOGNA
FACOLTA' DI SCIENZE MATEMATICHE, FISICHE E NATURALI
CORSO DI LAUREA IN SCIENZE DI INTERNET

STUDIO DEI FUNCTION POINT

Tesi di laurea in
Sistemi e Processi Organizzativi

Relatore

Prof. Paolo Ciancarini

Presentata da

Matteo Zucchini

Sessione III
Anno Accademico 2006/2007

INDICE

PREMESSA.....	pag. 7
1 L'ingegneria del software.....	pag. 9
2 Il Function Point.....	pag. 11
2.1 Perché è stato creato il Function Point.....	pag. 12
2.2 Funzionamento del Function point.....	pag. 15
2.2.1 Caratteristiche.....	pag. 15
2.2.2 Funzionamento.....	pag. 17
2.2.2.1 Individuare il tipo di conteggio.....	pag. 18
2.2.2.2 Identificare l'ambito di conteggio e il confine.....	pag. 19
2.2.2.3 Funzioni di tipo dati.....	pag. 20
2.2.2.4 Funzioni di tipo transazione.....	pag. 25
2.2.2.5 Calcolo del fattore di aggiustamento.....	pag. 32
2.2.3 Problemi d'applicazione.....	pag. 41
2.2.4 I Function Point e l'UML.....	pag. 41
3 Il COSMIC Full Function Point.....	pag. 43
3.1 Storia del COSMIC Full Function Point.....	pag. 43
3.2 Funzionamento COSMIC-FFP v. 2.2.....	pag. 44
3.2.1 Applicabilità del metodo.....	pag. 44
3.2.2 Estrazione dei requisiti funzionali.....	pag. 45
3.2.3 Fase di mappatura.....	pag. 48
3.2.3.1 Identificazione degli strati del software.....	pag. 48
3.2.3.2 Identificazioni dei confini del software.....	pag. 51
3.2.3.3 Identificazione dei processi funzionali.....	pag. 52
3.2.3.4 Identificazione dei gruppi di strati.....	pag. 54

3.2.3.5	Identificazione degli attributi di dati.....	pag. 56
3.2.4	Fase di misurazione.....	pag. 57
3.2.4.1	Identificazione dei movimenti di dati.....	pag. 58
3.2.4.2	Applicazione della funzione di misurazione.....	pag. 66
3.2.4.3	Aggregazione dei risultati della funzione di misurazione.....	pag. 67
3.2.5	Registrazione delle misurazioni COSMIC-FFP.....	pag. 68
3.2.5.1	Etichettatura dei risultati di misurazione COSMIC-FFP.....	pag. 68
3.2.5.2	Archiviazione dei risultati di misurazione COSMIC-FFP.....	pag. 69
4	Qualche passo indietro: le LOC.....	pag. 71
4.1	Definizione.....	pag. 71
4.2	Conteggio LOC.....	pag. 71
5	Esempio pratico: il gioco degli scacchi.....	pag. 73
5.1	Diagrammi UML.....	pag. 73
5.1.1	Struttura.....	pag. 73
5.1.2	Comunicazione.....	pag. 75
5.2	Conteggio con metodo IFPUG 4.2.1.....	pag. 77
5.2.1	Gli ILFs.....	pag. 78
5.2.2	Gli EIFs.....	pag. 79
5.2.3	Gli Els.....	pag. 79
5.2.4	Gli EOs.....	pag. 80
5.2.5	Gli EQs.....	pag. 81
5.2.6	Tabelle riepilogative.....	pag. 82
5.3	Conteggio con metodo COSMIC-FFP v. 2.2.....	pag. 83
5.3.1	Interfaccia: processi funzionali.....	pag. 83
5.3.2	Chess Model: processi funzionali.....	pag. 85

5.3.3	Tabella riepilogativa	pag. 86
6	Il conteggio delle SLOC	pag. 87
6.1	Il Backfiring	pag. 87
6.2	Conteggio con metodo LOC: Winboard	pag. 88
6.3	Conteggio con metodo LOC: Slow Chess	pag. 89
6.4	Conteggio con metodo LOC: Migoya Chess	pag. 90
6.5	Conteggio con metodo LOC: Java Chess	pag. 91
6.6	Confronti	pag. 92
6.6.1	Confronto tra programmi reali	pag. 92
6.6.2	Confronto tra programmi reali e specifiche	pag. 92
7	Considerazioni Personali	pag. 95
7.1	Sul metodo IFPUG 4.2.1	pag. 95
7.2	Sul metodo COSMIC-FFP v. 2.2	pag. 96
7.3	Sul metodo SLOC	pag. 97
7.4	Considerazioni finali	pag. 97
WEBBOGRAFIA		pag. 99
BIBLIOGRAFIA		pag. 101

PREMESSA

Fin dalla sua prima comparsa sulla terra, l'uomo ha creato metodi e strumenti per misurare e quantificare ciò che gli stava intorno e tramite queste misure, fare determinate valutazioni. Le unità di misura, al pari di ogni altra cosa nel Mondo, si sono evolute, modificate e adattate secondo le esigenze che l'uomo aveva: da sempre è stato necessario avere stime e misure per i liquidi, per quantificare i solidi presenti in una determinata area, oppure misurarne il peso. Già nel decimo secolo nei registri delle contee di Bologna usava determinare la dimensione dei campi con la nota unità di misura della tornatura. Un esempio di come l'uomo abbia sempre avuto necessità di determinare le dimensioni di ciò che aveva intorno e in questo, l'uomo, pur passando secoli e secoli, non è cambiato anzi: cerca di determinare sempre con più precisione tutto ciò che lo riguarda da vicino e anche il campo informatico è racchiuso nella sfera di quegli ambiti che l'uomo ha necessità di misurare. Basti pensare alle unità di misura per quantificare le informazioni contenute in un hard disk oppure la frequenza con cui un processore lavora per capire che anche il campo dell'informatica è strettamente correlato a quello delle unità di misura. Attualmente l'uomo ha ormai creato unità di misura per ogni cosa: forse non ce ne rendiamo conto ma le misurazioni sono ovunque e ci permettono di valutare tramite esse, rischi e opportunità, benefici, profitti, costi, ricavi ecc. Così, come per molte altre cose, anche per il software venne il momento di introdurre un'unità di misura, qualcosa che permettesse di valutare il lavoro dei programmatori e di quantificare anche economicamente gli sforzi profusi da team di sviluppo e ingegneri del software. Inizialmente si pensò di utilizzare il calcolo delle **LOC** (Lines Of Code, cap. 4) per la misurazione e la valutazione di un software ma questa unità di misura risultò avere numerose imperfezioni e impedimenti: basti pensare al fatto che si avevano numerose difficoltà a spostare le valutazioni da un linguaggio di programmazione all'altro rendendo difficoltoso e talvolta impossibile, far confronti diretti tra software simili. Così nel corso degli anni si è sviluppato un nuovo modo di valutare il software, il Function Point. L'argomento che si affronterà in questa tesi è appunto il Function Point, di cui si analizzeranno varianti, sviluppi e metodi di applicazione.

1. L'INGEGNERIA DEL SOFTWARE

Prima di entrare nel dettaglio e spiegare il funzionamento e l'utilizzo del function point, argomento di questa dissertazione, è bene definire cosa sia l'Ingegneria del Software con chiarezza. Così come recita l'IEEE Standard Glossary of Software Engineering, l'ingegneria del software è definita come:

“Applicazione di un approccio sistematico, disciplinato e quantificabile allo sviluppo, all'operatività e alla manutenzione del software”

In altri termini l'ingegneria del software è la disciplina tecnologica e gestionale che riguarda la produzione sistematica e la manutenzione dei prodotti software che sono sviluppati e modificati entro i tempi e i costi preventivati.

Il termine software è ormai entrato nel nostro linguaggio comune, questo perché ciò che è rappresentato da questo termine è intorno a noi, nella vita quotidiana delle persone, negli strumenti che utilizzano, siano essi oggetti per uso domestico o strumenti di lavoro. Dovendo chiarire il concetto di software si può citare una delle centinaia di definizioni presenti sul web:

“Il termine software (usato in ambito informatico) indica un programma o un insieme di programmi in grado di funzionare su un elaboratore”

Quindi, unendo le definizioni e i concetti espressi, si può dire che l'ingegneria del software produce, elabora e mantiene programmi, o un insieme di programmi, che permettono di far funzionare un elaboratore, più conosciuto come computer.

A dimostrazione del fatto che il software è entrato nelle nostre vite, basta pensare a quanti oggetti in casa nostra hanno un computer al proprio interno: caldaia, forno a microonde, frigoriferi sono tutti oggetti di uso quotidiano che hanno al loro interno un software. L'ingegneria del software sviluppa questi programmi e li rende utilizzabili anche all'utente inesperto semplificando la vita dell'uomo. Quello che però si andrà ad analizzare in questa tesi non è come siano sviluppati i software bensì come essi vengono valutati in termini di costo di produzione e quindi di prezzo. Queste sono solo due delle cose che il Function Point ci permette di osservare; nei capitoli successivi verrà definito questo importante strumento e verrà spiegato come il suo utilizzo stia prendendo sempre maggior consistenza nell'ingegneria del

software e quali sono le ragioni per cui alcune evoluzioni del Function Point, come il Cosmic Function Point, abbiano raggiunto il livello di standard ISO.

2. IL FUNCTION POINT

Ora passiamo a definire l'argomento principale della tesi: il Function Point. Citiamo qui di seguito diverse definizioni tratte dal web:

Mondomatica.it

“Function Point sono una metrica della dimensione funzionale di un'applicazione, basata sul numero e tipo delle informazioni in entrata, in uscita e memorizzazione”

cnipa.gov

“Il Function point è una misura standard della dimensione funzionale, misura che riduce l'astrattezza del prodotto, fornendo dei riferimenti precisi da valutare”

en.wikipedia.org

“A function point is a unit of measurement to express the amount of business functionality an information system provides to a user”

Il Function Point rientra quindi nel campo delle misure, della metrica, e come per le altre metriche, serve a dare una dimensione a ciò che abbiamo di fronte. Nello specifico il metodo dei Function Point è utilizzato sui software, citati nel capitolo precedente, consentendone una misurazione in base alle informazioni recepite in entrata, alle informazioni che vengono date in output e alle informazioni che vengono memorizzate.

Nella seconda definizione possiamo cogliere un modo diverso di vedere il Function Point: il Function Point *“riduce l'astrattezza del prodotto”*. Ciò significa che questa metrica ci permette di avere una valutazione più concreta di quel che è un software andando al di là dell'astrazione tipica di un'applicazione. Continua affermando che *“fornisce dei riferimenti precisi da valutare”* e questo è il punto più importante della definizione perché fa capire l'importanza di aver creato uno strumento in grado di dare riferimenti chiari, precisi e soprattutto, valutabili.

Così come evidenziato dalla definizione di software data dal glossario informatico di Manuali.it *“il software è qualcosa di intangibile”*; ciò spiega l'importanza di avere uno strumento di concretizzazione come il Function Point. Questo strumento

permette quindi di poter toccare con mano ciò che inizialmente non ci è possibile percepire. Vediamo un esempio.

Supponiamo di avere due figure di riferimento, un venditore e un compratore, che devono comunicare tra loro riguardo all'acquisto di un software. Il venditore propone un valore monetario per il prodotto, ma il nostro compratore è scrupoloso e vuole capire nel dettaglio cosa ha portato a fare questa valutazione alla controparte. In mancanza di un sistema dettagliato con regole ben definite, che stabiliscono come valutare il software, diventa impossibile per il venditore dimostrare che ciò che sta vendendo ha effettivamente quel valore.

Questa è una delle ragioni per cui il Function Point assume un'importanza notevole nella valutazione del software; vedremo più avanti nel dettaglio quali sono gli altri motivi che hanno spinto Alan Albrecht ad introdurre i Function Point nell'Ingegneria del Software.

2.1 Perché è stato creato il Function Point

Il Function Point è stato creato in risposta alle esigenze del manager di un progetto software di disporre di una valutazione qualitativa e quantitativa del processo di sviluppo:

"You can't control what you can't measure" – Tom DeMarco –

Questa citazione è molto esplicativa dei motivi per cui si è reso necessaria la creazione di un metodo per misurare un software: non possiamo gestire ciò che non possiamo misurare. E presa questa frase come un assoluto ci fa capire quanto sia importante poter misurare ed è qui che nasce il Function Point.

I motivi principali che hanno spinto Alan Albrecht a creare il Function Point possono essere riassunti nei punti seguenti:

- **Stimare** la produttività e quindi fare confronti fra le diverse tecnologie e metodi di sviluppo per scegliere e migliorare
- **Predire** la durata di un progetto e quindi il costo, utile sia per uso interno che per fare preventivi

- **Pianificare** il lavoro di sviluppo in base alle risorse
- **Controllare** che un progetto rispetti i tempi

Questi quattro riassumono quindi i motivi per cui vengono utilizzati sistemi di misura per valutare un software; queste valutazioni vengono effettuate su diversi tipi di entità:

- **Processi**, attività correlate allo sviluppo del software
- **Prodotto**, qualunque semilavorato o sistema prodotto durante lo sviluppo del software
- **Risorse**, personale, risorse hardware o software necessarie ai processi

Stima

Come detto in precedenza, il Function Point ci consente di fare confronti tra tecnologie diverse; questo ci permette di poter fare valutazioni chiare e precise su quale tecnologia è preferibile impiegare nella realizzazione di un progetto, quale metodo di sviluppo è meglio utilizzare per poter migliorare la qualità del nostro lavoro. Quindi la stima, non riguarda solamente i fattori fisici e tecnologici da impiegare, ma anche i fattori umani, le risorse: dare una stima significherà anche stimare il costo in termini di forza lavoro, necessari nello sviluppo e nella realizzazione di un progetto.

Predizione

L'utilizzo di uno strumento come il Function Point permetterà ai suoi utilizzatori di effettuare previsioni sulle tempistiche e quindi sui costi necessari per sostenere lo sviluppo del progetto. Questo punto è importante perché non solo permette di effettuare previsioni utili per chi lavora all'interno del progetto ma anche per chi si occupa di effettuare preventivi, rendendo chiaro per il cliente quali siano i costi e i tempi necessari per la realizzazione del lavoro.

Pianificazione

Questo terzo punto può essere visto come una conseguenza dei primi due; pianificare il lavoro è conseguente all'aver stimato le dimensioni del progetto, aver definito le tempistiche e l'impiego delle risorse. Non per questo è da ritenersi meno importante: la pianificazione del lavoro è molto importante ed errori in questo settore possono talvolta portare a ritardi sui tempi di consegna, errori in fase di sviluppo e mancanza di coesione all'interno dell'ambiente di lavoro, tra le persone che sviluppano il progetto.

Controllo

Il controllo è infine l'ultimo punto che spiega perché si è resa necessaria l'introduzione di sistemi di valutazione per il software: il rispetto dei tempi di consegna, dei tempi previsti per la realizzazione delle varie fasi del progetto consentono di valutare quanto la pianificazione, la stima e le previsioni fossero accurate e quanto il sistema di misura fosse preciso.

Conclusioni

In conclusione possiamo dire che la possibilità di misurare un software in ogni sua fase, dallo sviluppo al prodotto finito, porta ad un aumento della produttività, ad un miglioramento della qualità, sia del lavoro che del prodotto, ed infine ad un rispetto dei tempi di consegna, fattore di non poca importanza al giorno d'oggi.

2.2 Funzionamento del Function Point

2.2.1 Caratteristiche

Il Function Point è una delle metriche più antiche e tuttora diffuse nell'ambito della valutazione di progetti software. Ciò che lo rende insostituibile e superiore ad altri metodi può essere riassunto nei cinque punti seguenti:

- Fornisce un **parametro adimensionale**
- Misura la dimensione di un software in termini delle **funzionalità offerte** all'utente
- La misurazione si basa sul **disegno logico del software** espresso in una forma qualsiasi: specifiche in linguaggio naturale, schemi Entità-Relazione, diagrammi di Flusso dei Dati, ecc.
- E' **indipendente dall'ambiente tecnologico** in cui si sviluppa il progetto
- Consente **confronti fra differenti progetti** e organizzazioni

Parametro adimensionale

Citando Wikipedia:

“un gruppo adimensionale (o numero adimensionale o numero caratteristico) è una quantità che descrive un determinato sistema fisico, ed è un numero puro, senza alcuna unità fisica.”

Quindi il Function Point da una valutazione adimensionale, non dipendente da nessuna unità fisica, che descrive il valore di un software. I parametri adimensionali sono nel nostro passato e largamente utilizzati in ambito scientifico ed ingegneristico, basti pensare al numero di Laplace, al numero di Eulero o al numero di Avogadro.

Funzionalità offerte

La misura delle dimensioni di un software tramite Function Point, valuta le funzioni offerte all'utente: se un prodotto presenta un numero limitato di funzioni per l'utilizzatore finale, dovrà necessariamente essere valutato in maniera riduttiva rispetto ad un progetto che prevede un numero superiore di funzioni rivolte all'utente.

Disegno logico del software

Il disegno logico di un software esprime, in forme differenti, quella che è la struttura del progetto, quali sono le parti che lo compongono e come esse sono collegate tra loro. Ci sono diverse forme per mostrare come è strutturato un software, da specifiche in linguaggio naturale allo sviluppo di diagrammi Entità-Relazione strettamente correlati con il linguaggio UML. Il function point viene utilizzato proprio per valutare la complessità degli schemi logici. Vedremo in seguito come il Function Point sia legato all'UML e ai suoi diagrammi e schemi.

Indipendenza dall'ambiente tecnologico di sviluppo

Una delle cose più importanti che distingue il Function Point da altri metodi di conteggio come le LOC, è proprio la possibilità, grazie al suo modo di valutare i software, di essere scollegato dal linguaggio di programmazione utilizzato. Quindi indipendentemente dal linguaggio di programmazione utilizzato per sviluppare il software, il Function Point è in grado di stimare il costo di un progetto, cosa che ci porta direttamente al punto successivo.

Confronto tra progetti differenti

Questo è sicuramente il punto di forza del Function Point, la possibilità di fare confronti tra progetti differenti. La valutazione data dal Function Point, come detto in precedenza, è indipendente dal linguaggio di programmazione; per questo motivo è possibile confrontare progetti sviluppati in due ambienti diversi. Come vedremo nel paragrafo successivo, il Function Point utilizza un metodo di conteggio che da un certo peso al tipo di linguaggio utilizzato: linguaggi più complessi avranno valutazioni più alte mentre linguaggi più semplici avranno valutazioni inferiori.

2.2.2 Funzionamento

Prima di addentrarci nello spiegare come funzionano i Function Point è doveroso premettere che vi sono diversi metodi di conteggio alcuni dei quali sono diventati standard Iso, ma di questo parleremo in seguito. I metodi principali, utilizzati in maniera consistente a livello globale sono tre:

- **IFPUG v. 4.2.1**, molto diffuso negli Stati Uniti, evoluzione dell'originale
- Symons Mark II, molto diffuso in Inghilterra
- **Cosmic Full Function Points**, inventati da un gruppo guidato da un ricercatore canadese, molto utili per misurare sistemi real-time.

In questa dissertazione ci occuperemo però solo di IFPUG e Cosmic Full Function Points che sono entrambi conformi allo standard ISO; in particolare nello spiegare il funzionamento, faremo riferimento all'IFPUG v. 4.2.1.

Il metodo di cui ci occupiamo consiste nell'identificare cinque tipi di funzioni, o funzionalità:

Funzioni di Tipo Dati

- File interni logici
- File esterni di interfaccia

Funzioni di Tipo Transazione

- Input esterno
- Output esterno
- Interrogazioni esterne

Una volta identificate le funzioni, a ciascuna di esse si assegna un peso calcolato sulla base della quantità di dati e sulla complessità delle relazioni tra loro. La somma dei pesi di tutte le funzioni costituisce il **Numero di Function Points Non Pesato**.

Infine, questo numero è moltiplicato per un fattore di correzione ottenuto considerando un insieme di 14 **Caratteristiche Generali del Sistema**.

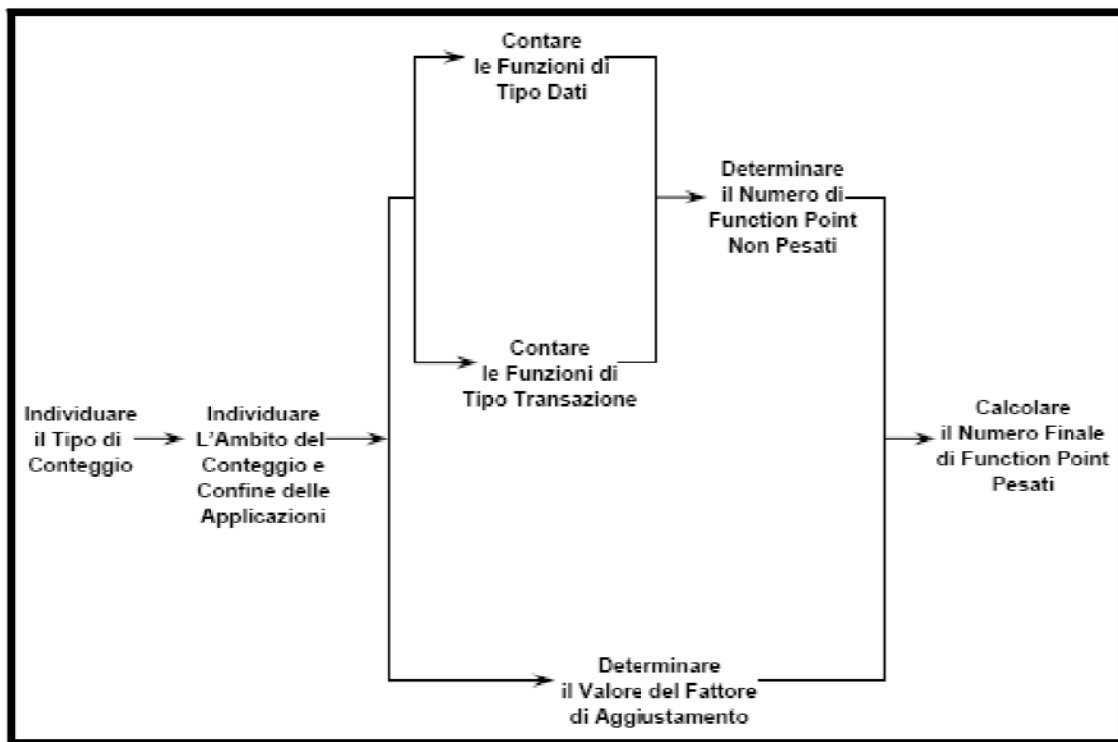


Figura 2.1 – Fasi di conteggio dei Function Point

Queste sono le fasi che è necessario affrontare per arrivare a determinare il numero di Function Point di un progetto software. Vedremo ora in dettaglio ognuno di questi step fino ad arrivare al numero finale di **Function Point Pesati**.

2.2.2.1 Individuare il tipo di conteggio

Per individuare il tipo di conteggio è ovviamente necessario conoscere preliminarmente quali sono i tipi che abbiamo a disposizione; esistono tre tipi di conteggio:

- *Per progetti di sviluppo*: in questo caso si procede al calcolo dei FP di un software da realizzare ex novo, più un eventuale conversione dei dati dalla vecchia applicazione

- *Per progetti di manutenzione evolutiva*: misura la modifica di un software esistente comprendendo funzioni aggiunte, modificate, cancellate e di conversione
- *Per un'applicazione esistente*: consente il calcolo dei Function Point cosiddetti *installati* e il loro aggiornamento. Questo punto comprende:
 - ✓ Calcolo dei Function Point iniziali il quale differisce dal calcolo per i progetti di sviluppo perché non prevede funzioni di conversione
 - ✓ Aggiornamento dei Function Point dopo ogni manutenzione evolutiva il quale differisce dal calcolo per un progetto di manutenzione evolutiva perché i punti delle funzioni cancellate sono sottratte invece che sommate.

Come si può notare, i tre tipi di conteggio coprono l'intera fase di vita di un progetto, dalla fase di sviluppo a quella di aggiornamento passando per la fase di manutenzione. Questo è un altro punto di forza del Function Point, la possibilità di essere applicato nell'intero ciclo di vita di un progetto software.

2.2.2.2 Identificare l'ambito del conteggio e il confine delle applicazioni

Identificare l'ambito del conteggio significa identificare le funzionalità che devono essere considerate in un conteggio. L'ambito stabilisce quali funzioni devono essere incluse nel conteggio le quali possono essere considerate in più di una applicazione.

Il confine invece, è la linea di separazione tra le applicazioni che si stanno misurando e le applicazioni esterne o l'utente.

Vi sono un paio di regole da rispettare per identificare il confine e l'ambito di conteggio:

- Il confine è determinato basandosi sul punto di vista dell'utente
- Il confine tra applicazioni collegate è basato su aree funzionali distinte dal punto di vista dell'utente e non in funzione degli aspetti tecnologici

Riguardo al confine è bene specificare che la presenza di più applicazioni che devono essere considerate nel conteggio porta ad un numero superiore di confini che dovranno quindi esser presi in considerazione.

2.2.2.3 Funzioni di Tipo Dati

In questa sezione abbiamo due tipi di file, i **file interni logici** ed i **file esterni di interfaccia**. Entrambi verranno definiti qui di seguito:

*“Un **ILF** (Internal Logic File) è un gruppo di dati logicamente collegati o di informazioni di controllo, riconoscibili dall'utente, mantenuti all'interno del confine dell'applicazione che si sta misurando.”* – tesionline –

In altre parole un ILF è un gruppo di dati o informazioni di controllo logicamente collegati e riconoscibili dall'utente che sono mantenuti all'interno dei confini dell'applicazione. Il compito primario di un ILF è di contenere dati mantenuti attraverso uno o più processi elementari dell'applicazione che si sta contando.

*“Un **EIF** (External Interface File) è un gruppo di dati logicamente collegati o di informazioni di controllo, riconoscibili dall'utente, mantenuti all'esterno del confine dell'applicazione che si sta misurando.”* – tesionline –

Quindi l'EIF è un gruppo di dati o informazioni di controllo logicamente collegati e riconoscibili dall'utente che sono referenziati dall'applicazione ma sono mantenuti all'interno dei confini di un'altra applicazione. Il compito primario di un EIF è di contenere dati referenziati da uno o più processi elementari dell'applicazione che si sta contando. Questo significa che un EIF contato per un'applicazione deve essere un ILF in un'altra applicazione.

Regole per l'identificazione di un gruppo ILF

Ogni gruppo di dati od informazioni di controllo che verifica le seguenti due regole, può essere considerato appartenente a questa categoria:

- Il gruppo di dati o informazioni di controllo è logico e identificabile dall'utente
- Il gruppo di dati è mantenuto all'interno del confine dell'applicazione che si sta contando da un processo elementare

Alcuni esempi possono essere:

- informazioni sugli impiegati, sui prodotti, sui clienti, ecc.
- registrazioni di prelievi da un conto corrente, di spese fatte con credit card, di movimentazione di magazzino, ecc.
- dati sulla sicurezza dell'applicazione (come password, accessi,..)
- dati di HELP
- dati di log (registrazione delle operazioni effettuate)

Regole per l'identificazione di un gruppo EIF

Ogni gruppo di dati od informazioni di controllo che verifica le seguenti regole, può essere considerato appartenente a questa categoria:

- Il gruppo di dati o informazioni di controllo è logico e identificabile dall'utente
- Il gruppo è referenziato dall'applicazione misurata ed è ad essa esterno
- Il gruppo di dati non è mantenuto dall'applicazione che si sta misurando
- Il gruppo di dati è mantenuto in un ILF di un'altra applicazione

Alcuni esempi possono essere:

- Dati su entità gestite da altre applicazioni
- Dati sulla sicurezza mantenuti all'esterno dell'applicazione
- Dati di HELP mantenuti all'esterno dell'applicazione
- Dati di log mantenuti all'esterno dell'applicazione

Ad ogni ILF ed EIF viene assegnato un valore di complessità funzionale che varia a seconda del numero di elementi di tipo dati (Data Element Type, **DET**) e dal numero di elementi di tipo record (Record Element Type, **RET**).

Ecco una definizione di **DET**:

“Un campo unico riconoscibile dall'utente, non ricorsivo. Il numero di Det è utilizzato per determinare la complessità di ogni tipo di funzione e il suo contributo al numero di Function Point.” – tesionline –

Regole di conteggio dei DET

Ci sono diverse regole da seguire per contare i DET:

1. Conta un DET per ciascun campo unico riconoscibile dall'utente e non ripetuto mantenuto o recuperato da un ILF o da un EIF attraverso l'esecuzione di un processo elementare.

Esempio:

in un ILF o EIF conta un DET per i 12 campi per il valore del budget mensile. Conta un DET aggiuntiva per identificare il mese applicabile

2. Quando due applicazioni mantengono e/o referenziano lo stesso ILF/EIF ma ciascuna mantiene/referenzia DETs separati, conta solo i DETs usati da ciascuna applicazione per calcolare la complessità dell'ILF/EIF
3. Conta un DET per ogni dato richiesto dall'utente per stabilire una relazione con un altro ILF o EIF

Esempio:

in una applicazione Risorse Umane le informazioni sugli impiegati sono mantenute in un ILF. Il nome del lavoro dell'impiegato è parte delle informazioni sull'impiegato. Questo DET è contato perché è necessario per collegare un impiegato con un lavoro che esiste nell'organizzazione. Questo elemento di tipo dati è chiamato chiave esterna.

4. Per contare i DET è necessario considerare il progetto logico (in termini di tabelle) dei dati, perché bisogna considerare anche le chiavi esterne.

Progetto Logico Semplificato:

- a) Trasforma ciascuna entità in una tabella
- b) Trasforma ciascuna relazione molti a molti in una tabella

- c) Trasforma una relazione uno a molti da A a B aggiungendo alla tabella per B:
 - a. Gli attributi della relazione
 - b. La chiave primaria di A

- d) Trasforma le relazioni uno a uno aggiungendo alla tabella per una delle due entità:
 - a. Gli attributi relazione
 - b. La chiave primaria dell'altra entità

Questo quadro riassuntivo ci mostra i passaggi per creare un progetto semplificato derivato da uno più ampio e complesso, rendendo il calcolo dei DET più semplice e chiaro.

Passiamo ora alla definizione di **RET**:

“Un sottogruppo di dati elementari di un ILF o EIF, riconoscibile dall'utente.”
– tesionline –

I RET, a differenza dei DET, possono essere di due tipi diversi, **obbligatori** od **opzionali**; durante un processo elementare che aggiunge o crea un'istanza dei dati, l'utente può usare zero o più sottogruppi opzionali e almeno un sottogruppo obbligatorio.

Regole di conteggio dei RET

1. Conta un RET per ciascun sottogruppo opzionale o obbligatorio di un ILF o EIF

2. Se non ci sono sottogruppi, conta il ILF o il EIF come un RET

In altre parole deve essere contato un RET per ogni entità e per ogni relazione con attributi nell'ILF.

Esempio:

In una applicazione Risorse Umane, un impiegato deve avere uno stipendio fisso oppure uno stipendio ad ore e può avere dei subordinati. In questo caso quindi ci sono tre RET:

- Info su Impiegato Stipendiato (campo obbligatorio)
- Info su Impiegato ad Ore (campo obbligatorio)
- Info su Subordinato (campo opzionale)

Cerchiamo ora di collegare quanto detto in precedenza con l'argomento principale della dissertazione ovvero il Function Point; per collegare ILF, EIF, DET e RET all'argomento è necessario passare per il calcolo della complessità, collegamento chiarificato nella tabella seguente.

	1-19 DET	20-50 DET	51 o più DET
1 RET	Bassa	Bassa	Media
2-5 RET	Bassa	Media	Alta
6 o più RET	Media	Alta	Alta

La complessità può assumere tre connotazioni: bassa, media o alta. La tabella qui sopra spiega in che modo valutare la complessità, in base al numero di RET e DET calcolati.

Per ottenere il numero di Function Point non pesati, si utilizza una tabella di conversione che trasforma la complessità in un determinato numero di Function Point.

Complessità Funzionale	ILF	EIF
Bassa	7	5
Media	10	7
Alta	15	10

Questa tabella è stata creata dallo stesso ideatore dei Function Point, Alan Albrecht, basandosi esclusivamente su discussioni e prove.

2.2.2.4 Funzioni di Tipo Transazione

In questa sezione abbiamo tre tipi di funzioni: **Input Esterni**, **Output Esterni** ed **Interrogazioni Esterne**. Qui di seguito le definizioni.

*“Un Input Esterno (External Input, **EI**) elabora dati o informazioni di controllo che provengono dall'esterno del confine dell'applicazione misurata.”* – tesionline –

Quindi l'EI è un processo elementare dell'applicazione che elabora dati o informazioni di controllo provenienti dall'esterno del confine dell'applicazione. Il compito principale di un EI è di mantenere uno o più ILFs o di modificare il comportamento del sistema.

*“Un Output Esterno (External Input, **EO**) produce dati o informazioni di controllo inviati all'esterno del confine dell'applicazione che si sta misurando.”* – tesionline –

L'EO quindi è un processo elementare dell'applicazione che manda dati o informazioni di controllo all'esterno del confine dell'applicazione. Il compito principale di un EO è di presentare informazioni all'utente attraverso una logica di processo diversa o in aggiunta al recupero di dati o informazioni di controllo. La logica di processo deve contenere almeno una formula matematica o calcolo, creare dati derivati, mantenere uno o più ILFs o modificare il comportamento del sistema.

*“Un'Interrogazione Esterna (External Inquiry, **EQ**) rappresenta una combinazione di input (richiesta) ed output (reperimento).”* – tesionline –

L'EQ è un processo elementare che manda dati o informazioni di controllo fuori dal confine dell'applicazione. Il compito principale di una EQ è di presentare informazioni all'utente attraverso il recupero di dati o informazioni di controllo da un ILF o EIF. La logica di processo non contiene formule matematiche o calcoli e non crea dati derivati. Nessun ILF è mantenuto durante l'elaborazione e il comportamento del sistema non è alterato.

La differenza principale tra le funzioni di tipo transazione, è il loro obiettivo primario (Primary Intent, **PI**). Le funzioni che possono diventare un obiettivo primario di EI, EO ed EQ sono identificate nella tabella sottostante, a cui segue una legenda che chiarifica i termini in essa utilizzati.

Complessità Funzionale	EI	EO	EQ
Alterare il comportamento del sistema	PI	F	N/A
Mantenere uno o più ILFs	PI	F	N/A
Presentare informazioni all'utente	F	PI	PI

PI = Obiettivo primario della funzione di tipo transazione (Primary Intent, **PI**)

F = Funzione della funzione di tipo transazione, non è un obiettivo primario (Function, **F**)

N/A = Funzione non consentita (Not Allowed, **N/A**)

Tutto ciò è utile per poter identificare di fronte a quale tipo di funzione transazione si è, e come comportarsi di conseguenza. La tabella seguente mostra in sintesi quali sono i passaggi da percorrere, per calcolare la quantità di Function Point non pesati relativi alle funzioni EI, EO ed EQ.

Passaggio	Azione
1	Identificare il processo elementare
2	Determinare l'obiettivo primario del processo elementare e classificarlo come EI, EO o EQ
3	Convalidare le regole di identificazione della funzione di transazione
4	Determinare la complessità della transazione
5	Determinare il numero di Function Point non pesati della transazione

1. Identificare il processo elementare

Per identificare il processo elementare è necessario osservare quali attività l'utente compie all'interno dell'applicazione. Le regole seguenti devono essere applicate al processo che si vuole identificare come processo elementare.

- Il processo è la più piccola unità di attività, significativa per l'utente
- Il processo è indipendente e lascia l'attività dell'applicazione in uno stato conforme.

2. Determinare l'obiettivo e classificare il processo

Per identificare ogni processo elementare è necessario determinare a quale descrizione esso appartiene ed usare le regole ad esso associate per poi identificare il tipo di funzione transazione.

Regole di conteggio EI

Tutte le regole devono essere applicate ai processi elementari per essere considerati come un unico evento di un input esterno. Una volta verificate queste regole si potrà considerare la funzione come un EI:

- Le informazioni di controllo sono ricevute dall'esterno del confine dell'applicazione
- Deve essere mantenuto almeno un ILF se i dati che entrano nel confine non sono informazioni di controllo che alternano lo stato del sistema.
- Per i processi identificati, deve essere verificata **una** delle seguenti affermazioni:
 1. Il processo logico è lo stesso di altri processi logici eseguiti da altri EI per l'applicazione
 2. Il gruppo di dati identificati deve essere diverso da quelli di altri EI per la medesima applicazione
 3. Gli ILFs o gli EIFs referenziati sono differenti dai file referenziati da altri EI nella medesima applicazione

Regole di conteggio condivise per EO ed EQ

Tutte le regole devono essere applicate ai processi elementari per essere considerati come un unico evento di un output esterno o una interrogazione esterna. Una volta verificate queste regole si potrà considerare la funzione come un EO o un EQ:

- La funzione invia dati o informazioni di controllo al di fuori dei confini dell'applicazione
- Per i processi identificati, deve essere verificata **una** delle seguenti affermazioni:
 1. Il processo logico è lo stesso di altri processi logici eseguiti da altri EO o EQ per la medesima applicazione
 2. Il gruppo di dati identificati deve essere diverso da quelli di altri EO ed EQ nella medesima applicazione

3. Gli ILFs o gli EIFs referenziati sono differenti dai file referenziati da altri EO ed EQ nella medesima applicazione

Regola addizionale per gli EO

In aggiunta alle regole appena elencate per identificare un processo come EO, è necessario che sia verificata **una** delle seguenti affermazioni:

1. l'esecuzione logica di un processo elementare deve contenere almeno una formula matematica o un calcolo
2. l'esecuzione logica di un processo elementare crea dati derivati
3. l'esecuzione logica di un processo elementare mantiene almeno un ILF
4. l'esecuzione logica di un processo elementare altera lo stato del sistema

Regola addizionale per gli EQ

In aggiunta alle regole appena elencate per identificare un processo come EO, è necessario che siano verificate **tutte** le seguenti affermazioni:

1. l'esecuzione logica di un processo elementare riporta dati o informazioni di controllo da un ILF o da un EIF
2. l'esecuzione logica di un processo elementare non deve contenere formule matematiche o calcoli
3. l'esecuzione logica di un processo elementare non deve creare dati derivati
4. l'esecuzione logica di un processo elementare non deve mantenere un ILF
5. l'esecuzione logica di un processo elementare non deve alterare lo stato del sistema

3. Calcolo della complessità della transazione

Il numero di EIs, EOs ed EQs e le relative complessità funzionali determinano il contributo delle funzioni transazione in termini di Function Point non pesati. Ad ogni EI, EO ed EQ identificato si assegna una complessità funzionale basata sul numero di Tipi di File Referenziati (file types referenced, **FTR**) e sul numero di Elementi di Tipo Dato (data element types, **DET**). Qui di seguito le definizioni:

“L’FTR è un ILF o un EIF letto o mantenuto da una funzione di tipo transazione.”
– tesionline –

Nello specifico quindi un FTR può assumere due forme:

- Un ILF letto e mantenuto da una funzione di tipo transazione
- Un EIF letto da una funzione transazione

“Il DET è un campo unico riconoscibile dall’utente, non ricorsivo. Il numero di DET è utilizzato per determinare la complessità di ogni tipo di funzione e il suo contributo al numero di Function Point.” – tesionline –

Qui di seguito verranno mostrate le regole che definiscono FTR e DET e determinano la complessità ed il contributo.

Regole FTR per gli EI

Le regole che seguono vengono applicate quando si contano gli FTR:

- Conta un FTR per ogni ILF mantenuto
- Conta un FTR per ogni ILF o EIF letti durante l’esecuzione di un EI
- Conta solo un FTR per ogni ILF che è sia mantenuto che letto

Regole DET per gli EI

Le regole che seguono vengono applicate quando si contano i DET:

- Conta un DET per ogni utente riconoscibile o un campo non ripetuto che entra o esce dal confine dell’applicazione ed è richiesto per completare un EI
- Non contare campi che sono riportati o derivati dal sistema e registrati su un ILF durante il processo elementare se i campi non attraversano il confine del sistema
- Conta un DET per la capacità di inviare un messaggio di responso del sistema fuori dal confine dell’applicazione per indicare che è stato riscontrato un errore durante il processo, confermare che il processo è completo o verificare che il processo può continuare.

- Conta un DET per l'abilità di specificare un azione che deve essere presa solo se ci sono metodi diversi per invocare il medesimo processo logico.

Regola FTR condivisa per EO ed EQ

- Conta un FTR per ogni ILF o EIF letto durante l'esecuzione di un processo elementare

Regole aggiuntive FTR per EO

- Conta un FTR per ogni ILF mantenuto durante l'esecuzione di un processo elementare
- Conta solo un FTR per ogni ILF che è sia mantenuto che letto durante il processo elementare

Regole DET condivise per EO ed EQ

- Conta un DET per ogni utente riconoscibile o un campo non ripetuto che entra dal confine dell'applicazione ed è richiesto per specificare quando, cosa e come i dati vengono riportati o generati dal processo elementare
- Conta un DET per ogni utente riconoscibile o un campo non ripetuto che esce dal confine
- Se un DET entra ed esce dal confine, deve essere contato solo una volta per il processo elementare
- Conta un DET per la capacità di inviare un messaggio di responso del sistema fuori dal confine dell'applicazione per indicare che è stato riscontrato un errore durante il processo, confermare che il processo è completo, o verificare che il processo può continuare
- Conta un DET per l'abilità di specificare un azione che deve essere presa solo se ci sono più metodi per invocare lo stesso processo logico
- Non contare campi che sono riportati o derivati dal sistema e registrati su un ILF durante il processo elementare se i campi non hanno attraversato il confine del sistema
- Non contare campi letterali come DETs

- Non contare impaginazioni variabili o stampe generate dal sistema

4. Complessità della transazione e calcolo dei Function Point non pesati

Le tabelle seguenti ci mostrano in che modo trasformare il numero di DETs ed FTRs in un numero function point:

Calcolo Complessità EI

	1-4 DET	5-15 DET	16 o più DET
0-1 FTR	Bassa	Bassa	Media
2 FTR	Bassa	Media	Alta
3 o più FTR	Media	Alta	Alta

Calcolo Complessità EO ed EQ

	1-5 DET	6-19 DET	20 o più DET
0-1 FTR	Bassa	Bassa	Media
2-3 FTR	Bassa	Media	Alta
4 o più FTR	Media	Alta	Alta

Calcolo Function Point per EI ed EQ

Complessità Funzionale	Function Point
Bassa	3
Media	4
Alta	6

Calcolo Function Point per EO

Complessità Funzionale	Function Point
Bassa	4
Media	5
Alta	7

Le tabelle ci hanno quindi permesso di ottenere il numero di function point non pesati partendo dalla quantità di FTRs e di DETs.

2.2.2.5 Calcolo del fattore di aggiustamento

Il fattore di aggiustamento è un valore che moltiplicato per il numero di Function Point non pesati ci permette di ottenere il numero definitivo di questi ultimi, portando così a termine la nostra procedura di conteggio.

Il **Fattore di Aggiustamento** (Value adjustment factor, **VAF**) è basato su 14 caratteristiche generali del sistema (GSCs) che spiegano e danno un valore alle funzionalità generali che devono essere contate in una applicazione.

Ogni caratteristica ha una propria dettagliata descrizione che consente di determinare il grado di influenza che quella caratteristica ha sul progetto; questo grado di influenza varia in una scala da 0 a 5, da una influenza nulla ad una influenza molto forte.

Quando applicato, il VAF da un aggiustamento ai Function Point non pesati con una variazione di $\pm 35\%$, variando quindi tra 0,65 e 1,35.

Procedura per determinare il VAF

Qui di seguito sono mostrate le procedure per determinare il valore del fattore di aggiustamento:

Passaggio	Azione
1	Determinare il grado di Influenza (DI) di ogni caratteristica generale del sistema
2	Sommare i gradi di influenza ottenendo il grado totale (TDI)
3	Usare il TDI all'interno dell'equazione $VAF = (TDI * 0,01) + 0,65$
4	Determinare la complessità della transazione
5	Determinare il numero di Function Point non pesati della transazione

Caratteristiche generali del sistema

1. Comunicazione dati
2. Distribuzione dell'elaborazione
3. Prestazioni
4. Utilizzo intensivo della configurazione
5. Frequenza delle transazioni
6. Inserimento dati interattivo
7. Efficienza per l'utente finale
8. Aggiornamento interattivo
9. Complessità elaborativa
10. Riusabilità
11. Facilità di installazione
12. Facilità di gestione operativa
13. Molteplicità di siti
14. Facilità di modifica

Ora verranno spiegate in dettaglio ognuna di queste caratteristiche e il modo di valutarle per definirne il grado di influenza per arrivare al conteggio finale dei Function Point.

1. Comunicazione dati

La comunicazione dati descrive il grado con cui l'applicazione comunica direttamente con il processore.

Punteggio	Descrizione
0	L'applicazione non interagisce con altri programmi o è stand-alone.
1	L'applicazione non interagisce con altri programmi ma ha dati remoti in entrata o stampanti remote.
2	L'applicazione non interagisce con altri programmi ma ha dati remoti in entrata e stampanti remote.
3	L'applicazione include una collezione di dati online o una interfaccia TP (teleprocesso, TP) che la collega ad un programma indipendente o ad una richiesta di sistema.
4	L'applicazione è più di un interfaccia, ma supporta solo un tipo di protocollo TP di comunicazione.
5	L'applicazione è più di un interfaccia e supporta più di un tipo di protocollo TP di comunicazione.

2. Distribuzione dell'elaborazione

La distribuzione dell'elaborazione descrive il grado con cui l'applicazione trasferisce i dati tra i componenti dell'applicazione

Punteggio	Descrizione
0	L'applicazione non supporta il trasferimento dati o funzioni di processo tra i componenti del sistema
1	L'applicazione prepara i dati per l'utilizzo da parte dell'utente su un altro componente del sistema sia esso un foglio di calcolo o un database
2	I dati preparati per il trasferimento vengono trasferiti e processati su un altro componente del sistema
3	Il processo di distribuzione e il trasferimento dati sono online ed in una sola direzione
4	Il processo di distribuzione e il trasferimento dati sono online in entrambe le direzioni
5	Le funzioni di processo sono eseguite dal componente più appropriato del sistema

3. Prestazioni

Questa parte descrive il grado con cui le performance, del tempo di risposta e della capacità di comunicazione, sono influenzate dallo sviluppo dell'applicazione.

Punteggio	Descrizione
0	Nessuna richiesta speciale di performance è data dall'utente
1	Le performance e i requisiti di interfaccia sono dati e rivisti ma non è richiesta nessuna azione speciale
2	Il tempo di risposta o la capacità di comunicazione sono critiche durante le ore di picco. Non è richiesta alcuna interfaccia speciale per l'utilizzo della CPU. La fine del processo è per il giorno successivo.
3	Il tempo di risposta o la capacità di comunicazione è critico durante tutte le ore di lavoro. Non è richiesta alcuna interfaccia speciale per l'utilizzo della CPU. E' prevista la costruzione di un interfaccia di sistema per gestire la conclusione del processo.
4	In aggiunta, le richieste di performance sono più severe e richiedono la progettazione di un sistema di analisi delle performance
5	In aggiunta, il tool per l'analisi delle performance è usato in fase di progettazione, rilascio e/o fase di implementazione per venire incontro alle richieste delle utente.

4. Utilizzo intensivo della configurazione

L'utilizzo intensivo della configurazione descrive il grado con cui le restrizioni delle risorse del computer sono influenzate dallo sviluppo dell'applicazione.

Punteggio	Descrizione
0	Nessuna restrizione, implicita o esplicita è inclusa
1	Presenza di restrizioni ma facilmente gestibili dall'applicazione.
2	Sono incluse considerazioni di sicurezza o di tempo
3	Sono incluse specifiche richieste di processo per una specifica parte dell'applicazione.
4	Le operazioni di base richiedono costruzioni speciali sull'applicazione, nel processore centrale o in quello dedicato
5	In aggiunta ci sono costruzioni speciali sull'applicazione nei componenti distribuiti del sistema

5. Frequenza delle transazioni

La frequenza delle transazioni descrive il grado con cui il tasso delle transazioni d'affari sono influenzate dallo sviluppo dell'applicazione.

Punteggio	Descrizione
0	Nessun picco di transazione è anticipato.
1	Picco di transazione (mese, semestre, stagione, annualità) anticipato.
2	Picco di transazione settimanale anticipato.
3	Picco di transazione giornaliero anticipato.
4	Alto tasso di transazione stabilito dall'utente nei requisiti dell'applicazione o i livelli di servizio accordati sono così elevati da richiedere un processo d'analisi.
5	In aggiunta, è richiesto l'utilizzo di un tool di analisi delle performance durante le fasi di design, distribuzione e/o installazione.

6. Inserimento dati interattivo

L'inserimento di dati interattivo descrive il grado con cui i dati sono inseriti attraverso transazioni interattive.

Punteggio	Descrizione
0	Tutte le transazioni sono processate dall'applicazione.
1	1% - 7% di transazioni sono inserimento dati interattivi.
2	8% - 15% di transazioni sono inserimento dati interattivi.
3	16% - 23% di transazioni sono inserimento dati interattivi.
4	24% - 30% di transazioni sono inserimento dati interattivi.
5	Più del 30% delle transazione sono inserimento dati interattivi.

7. Efficienza per l'utente finale

L'efficienza per l'utente finale descrive il grado di considerazione per i fattori umani e facilita l'uso per l'utente che utilizza l'applicazione. Vi sono diverse funzioni che devono essere considerate:

- Strumenti di navigazione
- Menù
- Aiuto Online e Documentazione
- Movimento automatizzato del cursore
- Scorrimento

- Stampante remota disponibile tramite collegamento online
- Funzioni chiave prestabilite
- Lavori autonomi sottoscritti da transazioni online
- Selezione con cursore dei dati su schermo
- Uso intensive di indicatori come evidenziatori, sottolineature e immagini.
- Copia consistente della documentazione riguardante transazioni online
- Interfaccia Mouse
- Finestre di pop-up
- Tanti schermi quanti necessari per agevolare le funzioni d'affari
- Supporto a due lingue (contato come 4 oggetti)
- Supporto multilingua (contato come 6 oggetti)

Punteggio	Descrizione
0	Nessuna funzione.
1	1 - 3 funzioni.
2	4 - 5 funzioni.
3	6 o più funzioni ma non ci sono altri requisiti specifici relativi all'efficienza.
4	6 o più funzioni e ulteriori requisiti per migliorare l'efficienza per l'utente finale
5	6 o più funzioni e ulteriori requisiti per migliorare l'efficienza per l'utente finale utilizzando strumenti che dimostrino il raggiungimento degli obiettivi

8. Aggiornamento interattivo

L'aggiornamento interattivo descrive il grado con cui gli ILF sono aggiornati online.

Punteggio	Descrizione
0	Nessun aggiornamento.
1	Aggiornamento da 1 a 3 file inclusi. Volume di aggiornamento ridotto e recupero semplice.
2	Aggiornamento di più di 4 file inclusi. Volume di aggiornamento ridotto e recupero semplice.
3	E' incluso l'aggiornamento dei principali ILF.
4	In aggiunta, la protezione contro la perdita di dati è essenziale e deve essere progettata e programmata nel sistema.
5	In aggiunta, gli alti volumi portano considerazioni di costo nel processo di recupero. Elevata automazione nelle procedure di recupero con intervento minimo da parte degli operatori.

9. Complessità elaborativa

La complessità elaborativa descrive il grado con cui i processi logici influenzano lo sviluppo dell'applicazione.

Questi i componenti da tenere presente:

- Controlli sensibili e/o applicazioni con specifici processi di sicurezza
- Processo logico estensivo
- Processo matematico estensivo
- Molte eccezioni del processo in transazioni incomplete che richiedono di essere processate nuovamente
- Processi complessi per fornire più possibilità di input e output

Punteggio	Descrizione
0	Nessun componente.
1	Un componente.
2	Due componenti.
3	Tre componenti.
4	Quattro componenti.
5	Tutti e cinque i componenti.

10. Riusabilità

La riusabilità descrive il grado con cui l'applicazione ed il codice dell'applicazione devono essere progettati, sviluppati e supportati per essere usati in altre applicazioni.

Punteggio	Descrizione
0	Codice non riutilizzabile.
1	Il codice riusabile è usato con l'applicazione.
2	Meno del 10% dell'applicazione, considera necessario più di un utente.
3	Il 10% o più dell'applicazione, considera necessario più di un utente.
4	L'applicazione è strutturata e documentata per facilitare il riutilizzo e l'applicazione è personalizzata dall'utente tramite il codice sorgente.
5	L'applicazione è strutturata e documentata per facilitare il riutilizzo e l'applicazione è personalizzata per l'uso dall'utente tramite il mantenimento di specifici parametri.

11. Facilità di installazione

La facilità di installazione descrive il grado con cui la conversione da precedenti ambienti è influenzata dallo sviluppo dell'applicazione.

Punteggio	Descrizione
0	Nessuna considerazione speciale da parte dell'utente e nessun setup richiesto per l'installazione.
1	Nessuna considerazione speciale da parte dell'utente ma è richiesto un setup per l'installazione.
2	Requisiti di conversione ed installazione sono stabiliti dall'utente e per entrambi sono rilasciate guide di supporto. L'impatto con la conversione sul progetto non è considerato importante.
3	Requisiti di conversione ed installazione sono stabiliti dall'utente e per entrambi sono rilasciate guide di supporto. L'impatto con la conversione sul progetto è considerato importante.
4	In aggiunta al 2 suddetto, la conversione automatica e il tool per l'installazione sono forniti e testati.
5	In aggiunta al 3 suddetto, la conversione automatica e il tool per l'installazione sono forniti e testati.

12. Facilità di gestione operativa

La facilità di gestione operativa descrive il grado con cui l'applicazione assiste agli aspetti operativi come le fasi di avvio, di backup e recupero del processo.

Punteggio	Descrizione
0	Nessuna operazione speciale, in aggiunta alla normale procedura di backup, è impostata dall'utente.
1-4	Uno, più o tutti dei seguenti oggetti si applicano al progetto. Ogni oggetto ha valore uno, eccetto se è specificato altro: <ul style="list-style-type: none"> - Fornito sistema di avvio, di backup e di recupero del processo ma è richiesto l'intervento di un operatore - Fornito sistema di avvio, di backup e di recupero del processo e non è richiesto l'intervento di un operatore (valore 2) - L'applicazione minimizza il bisogno di montare il nastro. - L'applicazione minimizza il bisogno di maneggiare fogli
5	L'applicazione è realizzata per supportare operazioni inattese. Significa che non è necessario l'intervento di un operatore tranne che per avviare e spegnere l'applicazione. Il recupero automatico dagli errori è una caratteristica dell'applicazione.

13. Molteplicità di siti

La molteplicità di siti descrive il grado con cui l'applicazione è stata sviluppata in diverse locazioni e organizzazioni di utenti.

Punteggio	Descrizione
0	I requisiti utente non prevedono il bisogno di più di uno utente/sito di installazione.
1	Il bisogno di siti multipli è considerato nella progettazione e l'applicazione è progettata per operare solo su hardware e ambiente software, identici.
2	Il bisogno di siti multipli è considerato nella progettazione e l'applicazione è progettata per operare solo su hardware e/o ambienti software, simili.
3	Il bisogno di siti multipli è considerato nella progettazione e l'applicazione è progettata per operare solo su hardware e/o ambienti software, differenti.
4	Documentazione e piani di supporto sono forniti e testati per supportare applicazioni su siti multipli, descritte da 1 o 2.
5	Documentazione e piani di supporto sono forniti e testati per supportare applicazioni su siti multipli, descritte da 3.

14. Facilità di modifica

La facilità di modifica descrive il grado con cui l'applicazione è stata sviluppata per modificare facilmente il processo logico o la struttura dati.

Le caratteristiche seguenti possono essere applicate al progetto:

- Domande flessibili e facilità di rapporto sono fornite e possono gestire semplici richieste. (vale come 1 oggetto)
- Domande flessibili e facilità di rapporto sono fornite e possono gestire richieste di livello medio. (vale come 2 oggetti)
- Domande flessibili e facilità di rapporto sono fornite e possono gestire richieste complesse. (vale come 3 oggetti)
- Il controllo di dati degli affari è redatto in tabelle e gestito dall'utente tramite processi interattivi online, ma i cambiamenti hanno effetto solo nel giorno d'affari successivo. (vale come 1 oggetto)

- Il controllo di dati degli affari è redatto in tabelle e gestito dall'utente tramite processi interattivi online, ma i cambiamenti hanno effetto immediato. (vale come 2 oggetti)

Punteggio	Descrizione
0	Nessun componente.
1	Totale di un oggetto.
2	Totale di due oggetti.
3	Totale di tre oggetti.
4	Totale di quattro oggetti.
5	Totale di cinque oggetti.

2.2.3 Problemi d'applicazione

I due principali problemi di applicazione per quanto riguarda i Function Point possono essere riassunti nei seguenti due punti:

- Il calcolo non è semplice e deve essere fatto da esperti e con strumenti adeguati
- Principalmente applicati in software di tipo gestionale ove la parte dati e funzioni è sufficiente per descrivere i requisiti del software e il software.

E' chiaro che i Function Point possono essere applicati anche su progetti non inclusi nella categoria dei programmi gestionali ma è altresì chiaro che il progetto che si va a studiare deve dare un notevole numero di informazioni molto dettagliate affinché un esperto, con strumenti adeguati, possa essere in grado di fare questo tipo di valutazione.

2.2.4 I Function Point e l'UML

Il Function Point risulta essere uno strumento molto importante anche per la fase di progettazione: uno dei problemi in cui s'incorre in fase di formulazione requisiti è di capire l'entità delle cose da fare in termini di Function Point; sarebbe, quindi, comodo potere già avere alla stesura dei requisiti un'indicazione di massima nel numero minimo di Function Point non pesati, senza dover impiegare molto tempo nel conteggio e che tenga conto di un minimo di errore di valutazione.

I casi d'uso, parte del linguaggio UML, esprimono, in modalità grafica, tutti i requisiti in gioco e che possono costituire, grazie all'UML, un linguaggio comune tra fornitore e cliente, insieme alla metodologia IFPUG dei Function Point.

I problemi dei casi d'uso sono di non poca rilevanza, in quanto esprimono solo i requisiti funzionali (non si fa riferimento ai requisiti non funzionali) e non sono dettagliati come si dovrebbe.

Questa semplicità del caso d'uso può essere eccessiva e non aiuta, per mancanza di dettagli, un conteggiatore di Function Point che risulta esterno alla problematica del progetto; tuttavia, una valutazione di massima degli UFP minimi da produrre sarebbe molto d'aiuto per le pianificazioni, l'allestimento dello staff e la previsione dei costi.

Anche se i casi d'uso vengono analizzati dal conteggiatore, la loro validazione da parte del cliente non è garanzia di avere completamente analizzato e previsto ogni cosa, questo proprio per la mancanza di completezza dei casi d'uso dell'UML.

L'UML ha però diversi altri grafici molto più dettagliati e complessi, che a posteriori possono aiutare il conteggiatore a fare valutazioni molto più precise rispetto all'uso esclusivo dei casi d'uso.

3. IL COSMIC FULL FUNCTION POINT

Questa terza parte della dissertazione è incentrata sulla nascita, lo sviluppo e la storia del Cosmic Full Function Point. Vedremo quindi come è nato, come si è sviluppato e quali sono le differenze dal metodo standardizzato dell'IFPUG oltre a fornire una descrizione dei vantaggi e degli svantaggi nell'utilizzo di questa tecnica di misurazione del software.

3.1 Storia del COSMIC Full Function Point

Riprendendo quanto detto in precedenza, l'analisi dei Function Point è un tipico esempio di metodo che permette di misurare la dimensione di un software; questo metodo ha però in se un percorso evolutivo ben preciso che ha portato alla formazione di diversi gruppi di lavoro che a loro volta hanno creato metodi, tra loro differenti, ma rispondenti a determinate caratteristiche racchiuse nello standard ISO/IEC 14143-1: 2007.

Uno di questi gruppi è il COSMIC Group (the **C**ommon **S**oftware **M**easurement **I**nternational **C**onsortium) che dopo aver raccolto nel 1998 il lavoro svolto dal FFP Group (Full Function Point, **FFP** v1.0) l'anno precedente, si unisce a quest'ultimo per realizzare una nuova versione del metodo che diventa di dominio pubblico nell'ottobre del 1999 con il nome di COSMIC-FFP v2.0.

La necessità di questo gruppo di creare un proprio metodo di misurazione del software è dovuto al fatto che non si riteneva la prima versione, in grado di offrire valutazioni complete per software in real-time. Da qui l'unione dei due gruppi e la creazione di una seconda versione più adatta alle esigenze di chi doveva misurare questo tipo di software.

A breve verrà rilasciata una nuova versione, la terza, il cui nome è stato semplificato in COSMIC v3.0 e che introduce ulteriori raffinamenti alla versione precedente del sistema di misurazione; l'ultima versione è la COSMIC-FFP v2.2 di cui di seguito sarà spiegato il funzionamento.

3.2 Funzionamento COSMIC-FFP v 2.2

3.2.1 Applicabilità del metodo

Il metodo di misurazione COSMIC-FFP è progettato per essere applicabile a software dei seguenti domini:

- Software applicativo aziendale, tipicamente richiesto in supporto alla gestione dei processi aziendali, come il software bancario, assicurativo, per la contabilità, il personale, gli acquisti, la distribuzione o la produzione.
- Software real-time, avente il compito di seguire o governare eventi che accadono nel mondo reale.
- Ibridi di quanto sopra, come per esempio nei sistemi di prenotazione in tempo reale per linee aeree o alberghi.

Il metodo di misurazione COSMIC-FFP comporta l'applicazione di un insieme di modelli, regole e procedure a una data porzione di software così come essa è percepita nell'ottica dei suoi Requisiti Utente Funzionali (Functional User Requirements, **FUR**). Il risultato dell'applicazione di questi modelli, regole e procedure è un valore di una quantità numerica che rappresenta la dimensione funzionale del software misurata in base ai suoi FUR.

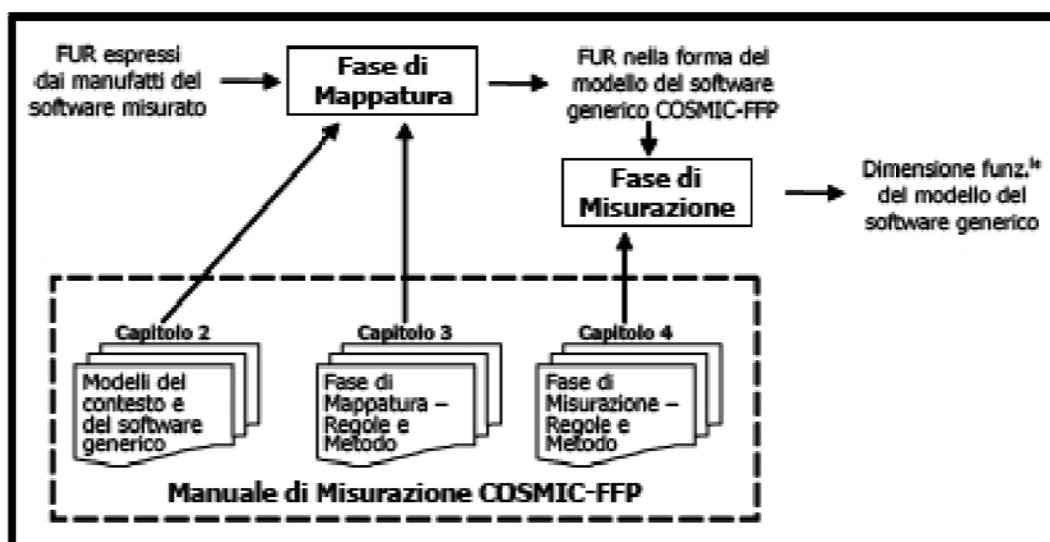


Figura 3.1 – Fasi di misurazione COSMIC-FFP

Il metodo di misurazione COSMIC-FFP è progettato per essere indipendente dalle decisioni implementative racchiuse nei manufatti operativi del software oggetto di misurazione. Per ottenere questa caratteristica, la misurazione è applicata ai FUR del software espressi nella forma del ‘modello del software generico’ COSMIC-FFP. Questa forma dei FUR è ottenuta tramite un processo di mappatura dei FUR espressi o impliciti nei manufatti concreti del software, illustrato qui sopra.

3.2.2 Estrazione dei requisiti utente funzionali

Molti sono gli aspetti del software. Nell’ottica del metodo di misurazione COSMIC-FFP, l’aspetto rilevante è costituito dalle ‘funzionalità’ che esso fornisce ai propri utenti, intendendo con ciò ‘le elaborazioni delle informazioni che il software deve compiere per conto degli utenti’. Le funzionalità fornite dal software ai propri utenti sono descritte tramite i FUR. Questi affermano ‘cosa’ il software deve fare per gli utenti ed escludono qualsiasi requisito tecnico o di qualità che dica ‘come’ il software deve operare.

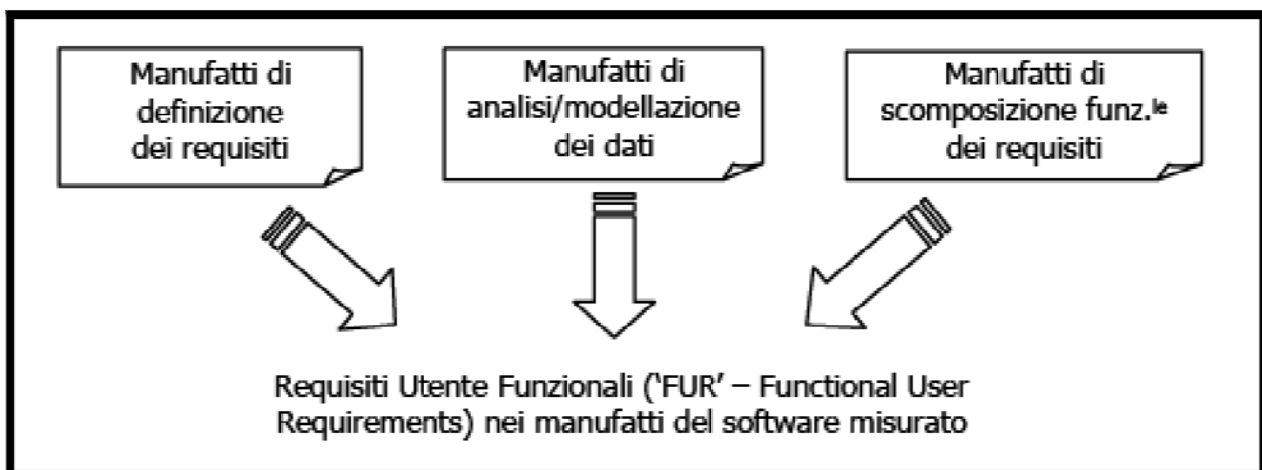


Figura 3.2 – Modello di FUR pre-implementazione

Come illustrato in Figura 3.2, i FUR possono essere derivati da manufatti dell’ingegneria del software che sono prodotti prima che il software sia stato realizzato, quindi un modello COSMIC-FFP dei FUR pre-implementazione. Così, la dimensione funzionale del software può essere misurata prima della sua implementazione in un sistema informatico.

In altre circostanze, il software potrebbe essere usato in assenza di, o con solo pochi, manufatti architetturali o di progettazione a disposizione, e i FUR potrebbero non

essere documentati. In tal caso, è ancora possibile derivare i FUR del software dai manufatti installati nel sistema informatico, anche successivamente alla sua implementazione, come illustrato nella figura sottostante che rappresenta un modello COSMIC-FFP dei FUR post-implementazione.

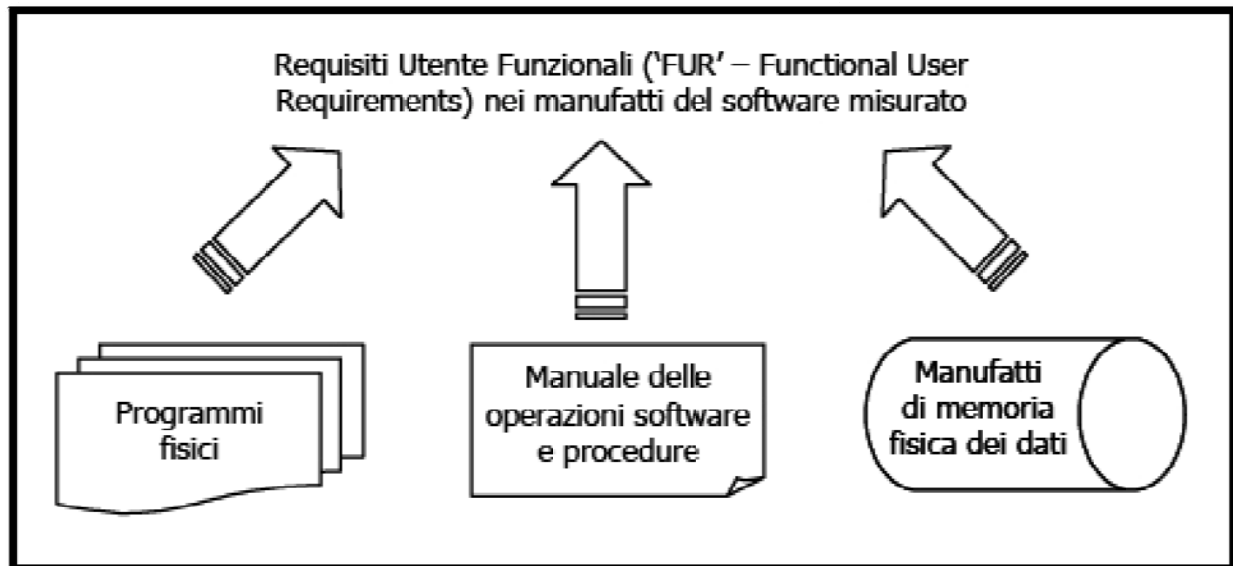


Figura 3.3 – Modello di FUR post-implementazione

Prima di intraprendere una misurazione con il metodo COSMIC-FFP è indispensabile definire attentamente lo Scopo, l'Ambito e il Punto di Vista della Misurazione. Ciò può essere considerato come il primo passo del processo di misurazione.

Scopo della misurazione

“Un'affermazione che definisce per quale motivo si intraprende la misurazione e/o per quale scopo sarà usato il risultato.”

Il metodo che qui è trattato definisce e descrive dimensioni, unità di misura e principi per la misurazione della dimensione funzionale del software. Esso fornisce anche alcune indicazioni per aiutare a individuare queste dimensioni in alcuni manufatti del software comunemente noti. In base allo scopo della misurazione, è compito del misuratore determinare i manufatti più appropriati da usare per effettuare una specifica misurazione.

Ambito della misurazione

“L’insieme dei FUR che devono essere inclusi in una specifica istanza di misurazione della dimensione funzionale.”

E’ importante definire l’ambito della misurazione, che si ricava dallo scopo della misurazione, prima di intraprendere una particolare misurazione. Per esempio, se lo scopo è misurare la dimensione funzionale del software consegnato da un particolare gruppo di progetto, sarà in primo luogo necessario definire i FUR di tutte le varie componenti che devono essere consegnate dal gruppo. Questi potrebbero includere i FUR del software usato una sola volta per convertire i dati del software che si intende sostituire. Se successivamente lo scopo è modificato in misurare la dimensione che li utenti avranno a disposizione una volta che il nuovo software sarà in funzione, esso risulterebbe essere più piccolo, poiché i FUR del software usato per la conversione non sarebbero inclusi nell’ambito della dimensione misurata.

Punto di vista della misurazione

“Un Punto di Vista dei FUR del software definito per scopi di misurazione della dimensione funzionale, dove ‘Punto di Vista’ è definito in ISO/IEC 10746-2 ‘Information technology – Open Distributed Processing – Reference Model: Foundations’) come:

*“Una forma di **astrazione** ottenuta mediante un insieme selezionato di concetti architetturali e di regole strutturali, al fine di concentrare l’attenzione su particolari aspetti di un software.”*

dove ‘astrazione’ è definito in ISO/IEC 10746-2 come:

“Il processo di soppressione dei dettagli irrilevanti per stabilire un modello semplificato, o il risultato di questo processo.””

E’ essenziale definire il punto di vista della misurazione, il quale di nuovo può discendere dallo scopo della misurazione. Il punto di vista della misurazione, in termini generali, determina il livello di dettaglio che può essere percepito e quindi misurato, entro l’ambito della misurazione. Il punto di vista della misurazione è altamente rilevante, poiché in generale misurazioni svolte da differenti punti di vista i misurazione non possono essere confrontate o sommate significativamente tra di loro.

3.2.3 Fase di mappatura

Il metodo di misurazione COSMIC-FFP valuta la misurazione della dimensione funzionale del software tramite due fasi distinte: la mappatura del software misurato sul modello del software generico COSMIC-FFP e la misurazione di specifici aspetti di questo modello del software generico. Questo paragrafo presenta le regole e il metodo della fase di mappatura. Il metodo generale per la mappatura del software sul modello del software generico COSMIC-FFP è riepilogato nella Figura 3.4.

3.2.3.1 Identificazione degli strati software

I FUR possono affermare esplicitamente, possono implicare, o il misuratore può dedurre, che essi riguardano software in differenti strati o differenti elementi di pari livello. Alternativamente, il misuratore può dover affrontare il dimensionamento di software esistente che si presenta distribuito in differenti strati o differenti elementi di pari livello. In entrambi i casi, si assuma che lo Scopo, l'Ambito e il Punto di Vista della Misurazione indichino che questi strati devono essere misurati separatamente. Per esempio, lo Scopo è la stima di progetto, dove gli strati saranno sviluppati con differenti tecnologie. È necessaria allora una guida per decidere se i FUR, o il software, comprendono uno o più strati o elementi di pari livello. Gli strati possono essere identificati attenendosi alla seguente definizione e ai seguenti principi.

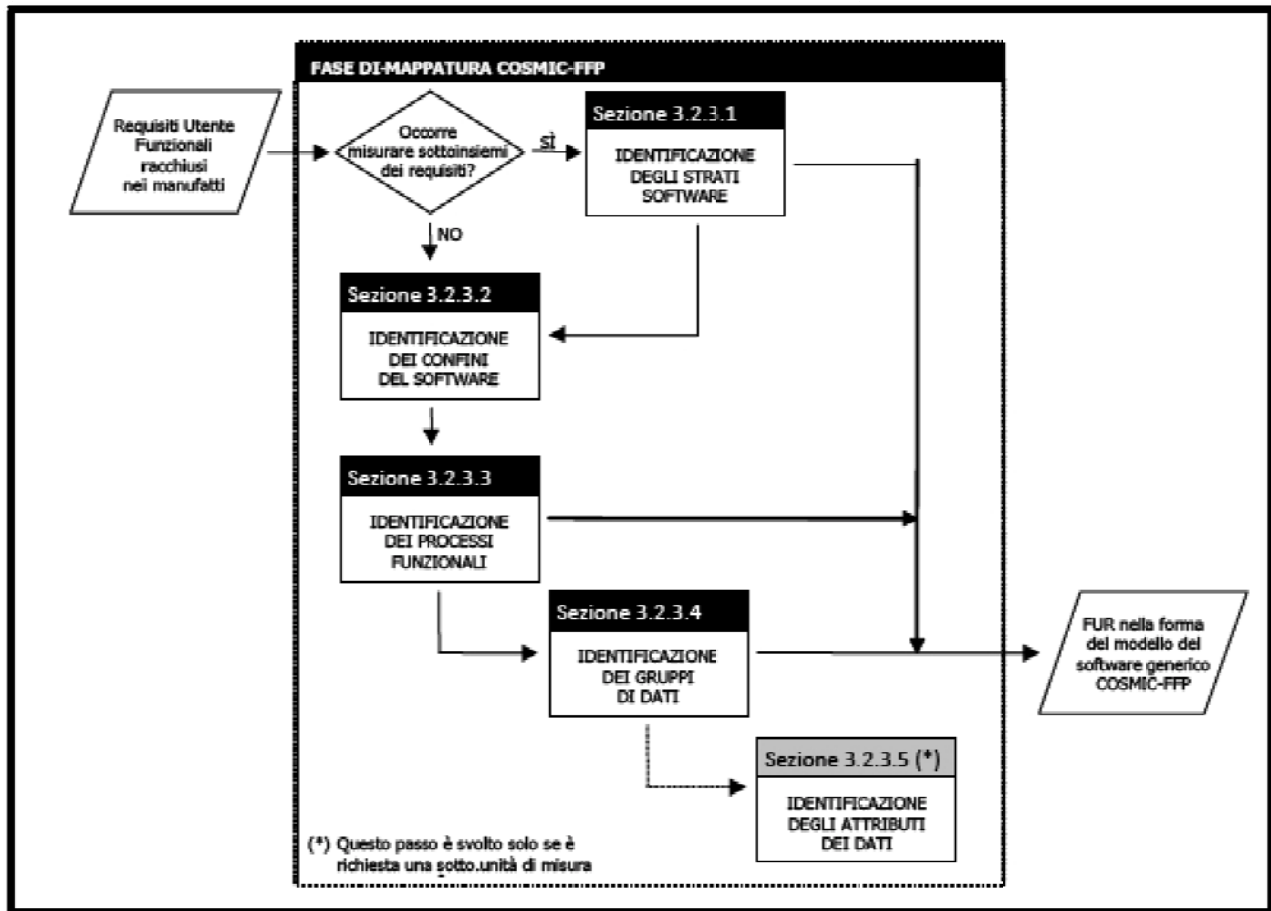


Figura 3.4 Metodo per la mappatura del software

Segue la definizione di **strato**:

“Uno strato è il risultato del partizionamento funzionale dell’ambiente software tale che tutti i processi funzionali in esso inclusi operano al medesimo livello di astrazione.

In un ambiente software multi-strato, i software appartenenti a due strati scambiano dati tra di loro tramite i rispettivi processi funzionali. Queste interazioni sono di natura gerarchica; considerandoli a coppie, uno strato è subordinato rispetto all’altro. Il software di uno strato subordinato fornisce servizi funzionali al software degli altri strati che usano i suoi servizi.”

L’identificazione degli strati è un processo iterativo. Gli strati esatti saranno raffinati man mano che il processo di mappatura procede. Una volta identificato, ogni candidato strato deve soddisfare i seguenti principi:

Principio	Descrizione
a	Il software di ogni strato fornisce funzionalità ai 'propri' utenti (un utente può essere un essere umano, un dispositivo fisico o un altro software, p.es. il software appartenente a un altro strato).
b	Il software di uno strato subordinato fornisce servizi funzionali al software appartenente a un altro strato che usa i suoi servizi.
c	Il software di uno strato subordinato potrebbe operare senza l'assistenza da parte del software appartenente allo strato che usa i suoi servizi.
d	Il software di uno strato potrebbe non operare correttamente se il software appartenente a uno strato subordinato dal quale esso dipende non sta operando correttamente.
e	Il software di uno strato non usa necessariamente tutte le funzionalità fornite dal software appartenente a uno strato ad esso subordinato.
f	In una gerarchia di strati, il software di un qualsiasi strato può essere subordinato rispetto a uno strato superiore al quale esso fornisce dei servizi.
g	I software appartenenti a uno strato e a uno strato ad esso subordinato possono fisicamente condividere e scambiare dati tra di loro. Nondimeno, il software di ogni strato interpreterà gli attributi dei dati in modo differente e/o li raggrupperà in differenti gruppi di dati.
h	Due software che condividono dati tra di loro non devono essere considerati appartenenti a strati differenti se interpretano in modo identico gli attributi dei dati che condividono.

E sottostare alle seguenti regole:

Regola	Descrizione
a	Pacchetti software di servizi funzionali, quali sistemi di gestione di database (DBMS – Database Management Systems), sistemi operativi o sistemi di gestione di dispositivi ('device drivers'), sono generalmente considerati come strati distinti.
b	Se il software è concepito usando un paradigma architetturale a strati prestabilito, come inteso in questa sede, allora si usi tale paradigma per identificare gli strati.
c	Il livello dell'applicazione software è generalmente considerato come il livello dello strato più alto.
d	In caso di dubbio, si usino i concetti di coesione e accoppiamento per fare distinzione tra strati interagenti.

Il concetto di strato software è un strumento utile per aiutare a distinguere i FUR allocati a differenti livelli di astrazione funzionale. Sono in uso molti modelli di architettura del software. Il modello a strati è usato in questa sede per fornire una vista funzionale del software. Altri modelli potrebbero essere usati, se forniscono, in parte o in forma completa, una vista funzionale del software.

3.2.3.2 Identificazione dei confini del software

Questo passo consiste nell'identificare il confine di ogni porzione del software misurato (a seconda del Punto di Vista della Misurazione, p.es. ogni strato o ogni elemento di pari livello appartenente a uno strato nel caso del Punto di Vista della Misurazione dello 'Sviluppatore').

E' necessario dare definizioni riguardo i termini "confine" e "utente":

*"Il **confine** è definito come un'interfaccia concettuale tra il software esaminato e i suoi utenti.*

Il confine di una porzione di software è la frontiera concettuale tra questa porzione e l'ambiente in cui essa opera, così come essa è percepita esternamente nell'ottica dei suoi utenti. Il confine permette al misuratore di distinguere, senza ambiguità, cosa è incluso nel software misurato da cosa è parte dell'ambiente operativo del software misurato."

"Un utente è definito come qualsiasi persona o cosa che comunica o interagisce con il software (oggetto di misurazione) in qualsiasi momento."

Per questa seconda definizione è doveroso fare un paio di precisazioni: nel manuale di Misurazione COSMIC-FFP, il termine 'utente' è ristretto al secondo dei due significati forniti da questa definizione, cioè "qualsiasi persona o cosa che comunica o interagisce con il software in qualsiasi momento" e gli utenti possono essere esseri umani, software o dispositivi ingegnerizzati.

L'identificazione dei confini è un processo iterativo. I confini esatti saranno oggetto di raffinamento man mano che il processo di mappatura procede. Una volta identificato, ogni candidato confine deve soddisfare il seguente principio:

Principio	Descrizione
a	Per definizione, esiste un confine tra ogni coppia identificata di strati, dove uno strato è utente dell'altro, e quest'ultimo è l'oggetto della misurazione. Analogamente, esiste un confine tra due qualsiasi distinte porzioni di software appartenenti al medesimo strato se esse scambiano dati tra di loro in comunicazioni 'tra pari livello' (o 'da pari a pari' – 'peer-to-peer'); in questo caso ogni porzione può essere un utente della porzione sua pari livello.

Le seguenti regole possono essere utili per convalidare lo stato di ogni candidato confine:

Regola	Descrizione
a	Si identifichino gli eventi d'innescò, quindi si identifichino i processi funzionali resi possibili da questi eventi. Il confine si colloca tra gli eventi d'innescò e questi processi funzionali.
b	Nel caso di software real-time o tecnico, si usi il concetto di strato come ausilio nell'identificazione del confine (sezione 3.2.3.1).

3.2.3.3 Identificazione dei processi funzionali

Questo passo consiste nell'identificare l'insieme dei processi funzionali del software misurato, in base ai suoi FUR.

Definizione di "processo funzionale" ed "evento d'innescò"

*“Un **processo funzionale** è una componente elementare di un insieme di FUR, contenente un insieme di movimenti di dati unico, compatto e indipendentemente eseguibile. Esso è innescato da uno o più eventi d'innescò o direttamente, o indirettamente tramite un **'attore'**. Esso è completato quando ha eseguito tutto ciò che si richiede di fare in risposta al (tipo di) evento d'innescò.*

*Un **'attore'** è un utente del sistema misurato, che agisce da intermediario per trasferire dati riguardanti un evento d'innescò al processo funzionale che deve rispondere a tale evento.”*

*“Un **evento d'innescò** è un tipo di evento che ha luogo all'esterno del confine del software misurato e avvia uno o più processi funzionali. Dato un insieme di FUR,*

ogni tipo di evento che innesca un processo funzionale è indivisibile per questo insieme di FUR.

Il clock e gli eventi di temporizzazione possono costituire eventi d'innescamento. Per quanto attiene al software, un evento o è avvenuto, o non lo è; esso è istantaneo.”

Una volta identificato, ogni candidato processo funzionale deve soddisfare i seguenti principi:

Principio	Descrizione
a	Un processo funzionale discende da almeno un FUR identificabile.
b	Un processo funzionale si avvia quando ha luogo un evento d'innescamento identificabile.
c	Un processo funzionale comprende almeno due movimenti di dati, un Entry più o un eXit o un Write.
d	Un processo funzionale appartiene a uno, e un solo, strato.
e	Un processo funzionale termina quando si raggiunge un punto di temporizzazione asincrona. Si raggiunge un punto di temporizzazione asincrona quando l'ultimo movimento di dati (finale) in una sequenza di movimenti di dati non è sincronizzato con un qualsiasi altro movimento di dati.

Le seguenti regole possono essere utili per convalidare lo stato di ogni candidato processo funzionale:

Regola	Descrizione
a	Per definizione, un (tipo di) evento d'innescamento dà luogo a un (tipo di) Entry d'innescamento, cioè al movimento di un (tipo di) gruppo di dati comprendente un certo numero di (tipi di) attributi dei dati. Se può aversi che certe occorrenze del gruppo di dati dell'Entry d'innescamento contengano solo dei sotto-insiemi dei valori di tutti gli (tipi di) attributi dei dati, allora l'eventuale esistenza di tali occorrenze non implica l'esistenza di un differente (tipo di) evento o processo funzionale.
b	Anche nel contesto del software real-time, un processo funzionale è innescato da un evento. Esso termina quando si raggiunge un punto di temporizzazione asincrona. Un punto di temporizzazione asincrona equivale a uno stato di attesa auto-indotto.

3.2.3.4 Identificazione dei gruppi dati

Questo passo consiste nell'identificare i gruppi di dati referenziati dal software misurato.

Definizione di “gruppo di dati” e “persistenza di un gruppo di dati”

*“Un **gruppo di dati** è un insieme distinto, non vuoto, non ordinato e non ridondante di attributi dei dati, in cui ogni attributo dei dati descrive un aspetto complementare del medesimo oggetto d'interesse. Un gruppo di dati è caratterizzato dalla propria **persistenza**.”*

*“La **persistenza di un gruppo di dati** è una caratteristica che descrive quanto a lungo il gruppo di dati è conservato nel contesto dei FUR. Si possono definire tre tipi di persistenza:*

***Temporanea:** Il gruppo di dati non sopravvive al processo funzionale che lo usa.*

***Breve:** Il gruppo di dati sopravvive al processo funzionale che lo usa, ma non sopravvive quando il software che lo usa cessa di operare.*

***Indefinita:** Il gruppo di dati sopravvive anche quando il software che lo usa cessa di operare.”*

La persistenza dei dati è una caratteristica usata per aiutare a distinguere tra i quattro tipi di sotto-processo, come spiegato nel capitolo 4. Una volta identificato, ogni candidato gruppo di dati deve soddisfare i seguenti principi:

Principio	Descrizione
a	Un gruppo di dati deve essere fisicamente realizzato all'interno del sistema informatico che supporta il software misurato.
b	Ogni gruppo di dati identificato deve essere unico e distinguibile attraverso il suo insieme unico di attributi dei dati.
c	Ogni gruppo di dati deve essere direttamente correlato a un oggetto d'interesse descritto nei FUR del software misurato.

La definizione e i principi di gruppo di dati sono volutamente generici, per risultare applicabili alla gamma più vasta possibile di software. Questa caratteristica ha uno svantaggio nel fatto che la loro applicazione alla misurazione di una specifica

porzione di software potrebbe essere difficoltosa. Di conseguenza, le seguenti regole, ricavate dalla pratica di misurazione della dimensione funzionale, possono aiutare nell'applicazione dei principi a casi specifici.

Regola	Descrizione
a	<p>APPLICAZIONE AL SOFTWARE APPLICATIVO AZIENDALE.</p> <p>La pratica di misurazione ha stabilito che, nel caso del software applicativo aziendale, si identifica un gruppo di dati per ogni 'tipo di entità' (o relazione in 'Terza Forma Normale' – TFN) del modello normalizzato dei dati del software misurato. Questi sono normalmente gruppi di dati che presentano persistenza indefinita e per i quali si richiede che il software memorizzi dati, riguardanti le entità (i tipi di entità) interessate.</p> <p>Inoltre, gruppi di dati che presentano persistenza temporanea sono creati ogniqualvolta vi sia un'interrogazione ad hoc che richiede dati circa qualche 'cosa' riguardo alla quale i dati non sono memorizzati con persistenza indefinita, ma possono essere derivati da (altri) dati memorizzati con persistenza indefinita. In tal caso, l'oggetto d'interesse temporaneo è l'oggetto del movimento di dati di tipo Entry dell'interrogazione ad hoc (i parametri di selezione per derivare i dati richiesti) e del movimento di dati di tipo eXit contenente gli attributi desiderati dell'oggetto d'interesse temporaneo.</p>
b	<p>APPLICAZIONE AL SOFTWARE REAL-TIME. La pratica di misurazione del software real-time ha stabilito che i gruppi di dati per questo tipo di software spesso assumono le seguenti forme:</p> <ul style="list-style-type: none"> • Movimenti di dati di tipo Entry provenienti dai dispositivi fisici tipicamente contengono dati riguardanti lo stato di un singolo oggetto d'interesse, come per esempio se una valvola è aperta o chiusa, o indicano un intervallo di tempo in cui i dati nella memoria volatile, di breve periodo, sono validi o non validi, o contengono dati che indicano che è avvenuto un evento critico e che causano un'interruzione. • Un commutatore di messaggi ('message-switch') può ricevere un gruppo di dati di messaggio come Entry e re-instradarlo immutato come eXit. Gli attributi del gruppo di dati di messaggio potrebbero essere, per esempio, 'mittente, destinatario, codice di instradamento e contenuto del messaggio', e il suo oggetto d'interesse è il 'messaggio'. • Struttura dati comune, che rappresenta oggetti d'interesse citati nei FUR, conservati nella memoria volatile e accessibili alla maggior parte dei processi funzionali del software misurato.

- Struttura dati di riferimento, che rappresenta grafici o tabelle di valori riportati nei FUR, conservati nella memoria persistente (nella memoria ROM, per esempio) e accessibili alla maggior parte dei processi funzionali del software misurato.
- Archivi di dati (file), comunemente denotati come 'file piatti' ('flat files'), che rappresentano oggetti d'interesse citati nei FUR, conservati in un dispositivo di memoria persistente.

3.2.3.5 Identificazione degli attributi dei dati

Questo passo consiste nell'identificare l'insieme degli attributi dei dati referenziati dal software misurato. In questa versione del metodo di misurazione, non è obbligatorio identificare gli attributi dei dati.

Definizione di "attributo dei dati" e "tipi di attributi dei dati"

*“Un **attributo dei dati** è la più piccola porzione di informazione, appartenente a un gruppo di dati, che trasporta un significato nell’ottica dei FUR del software.”*

*“Tra tutti gli attributi dei dati referenziati dal software misurato, si distinguono **tre tipi**. Due di essi sono validi allo scopo di misurare la dimensione funzionale del software e uno di essi non è valido per questo scopo.”*

Tipi di dati validi

Dati che descrivono o rappresentano il mondo dell'utente. Questo tipo di attributo dei dati ha un significato che è indipendente dal modo in cui è elaborato. Il nominativo e il salario di un impiegato hanno un significato indipendentemente da come saranno elaborati da un insieme di processi funzionali, che stamperebbero, per esempio, gli assegni delle paghe.

Dati contestuali. Questo tipo di dati specifica quali, quando o come altri dati devono essere elaborati dal software misurato. Il loro significato funzionale si comprende pienamente considerando il contesto in cui essi sono referenziati. Si consideri per esempio il seguente FUR: “produrre la stampa XYZ alle ore 10:00”. L'orario specificato (10:00) non trasmette il proprio significato funzionale indipendentemente dal contesto della 'stampa XYZ'. Esso determina quando deve essere prodotta la stampa.

Tipi di dati non validi

Dati tecnici di implementazione. Questo tipo di dati è creato solo perché è stata scelta una specifica tecnica di implementazione.

Una volta identificato, ogni candidato attributo dei dati deve soddisfare i seguenti principi:

Principio	Descrizione
a	Un attributo dei dati deve rappresentare un tipo di dati valido.
b	Un attributo dei dati deve rappresentare, nell'ottica dei FUR, la più piccola porzione di dati referenziata dal software misurato.

3.2.4 Fase di misurazione

Il metodo di misurazione COSMIC-FFP valuta la misurazione della dimensione funzionale del software tramite due fasi distinte: la mappatura del software misurato sul modello del software generico COSMIC-FFP e la misurazione di specifici aspetti di questo modello. Il calcolo della dimensione funzionale del software misurato è indipendente dall'impegno necessario per sviluppare o mantenere il software, dal metodo usato per sviluppare o mantenere il software e da qualsiasi componente fisica o tecnologica del software. La fase di misurazione è illustrata nella Figura 3.5.

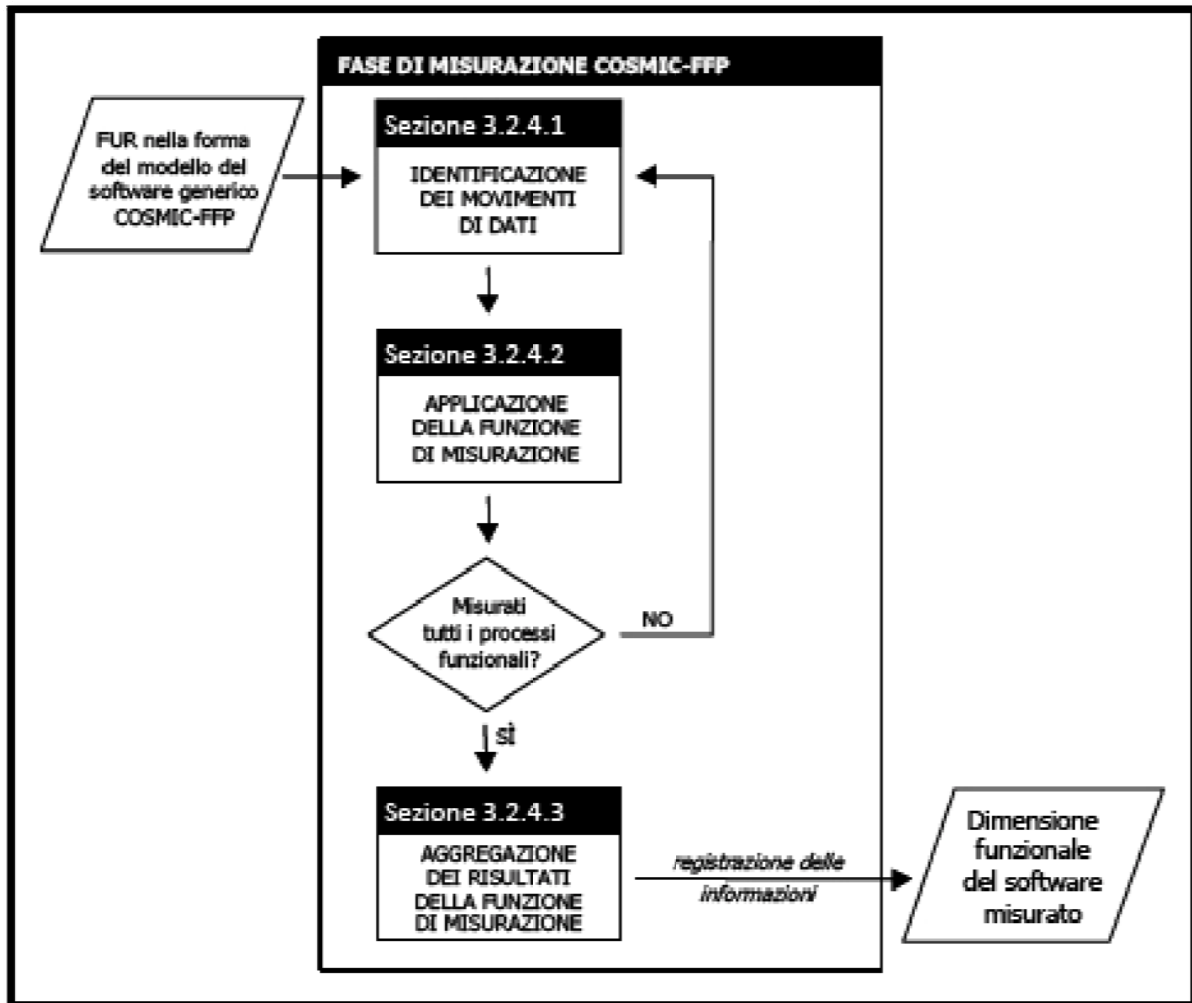


Figura 3.5 Fase di misurazione del software

3.2.4.1 Identificazione dei movimenti di dati

Questo passo consiste nell'identificare i tipi di sotto-processi di movimento di dati (Entry, eXit, Read e Write) di ogni tipo di processo funzionale. E' necessario dare alcune definizioni preliminari prima di passare alle regole di questa sezione.

Movimento di dati COSMIC-FFP

“Un movimento di dati COSMIC-FFP è una componente di un processo funzionale che muove uno o più attributi dei dati appartenenti a un unico gruppo di dati. Un movimento di dati COSMIC-FFP ha luogo nel corso dell'esecuzione di un processo funzionale. Esistono quattro sotto-tipi di movimento di dati COSMIC-FFP: Entry,

eXit, Read e Write, ognuno dei quali include una relativa specifica manipolazione dei dati.”

Entry (E)

“Un Entry (E) è un movimento di dati che muove un gruppo di dati proveniente da un utente attraverso il confine, verso il processo funzionale che lo richiede. Un Entry non aggiorna i dati che muove. Si noti anche che nel metodo COSMIC-FFP, un Entry è considerato includere certe manipolazioni dei dati associate ad esso (p.es. convalida dei dati inseriti).”

eXit (X)

“Un eXit (X) è un movimento di dati che muove un gruppo di dati da un processo funzionale attraverso il confine verso l'utente che lo richiede. Un eXit non legge i dati che muove. Si noti anche che nel metodo COSMIC-FFP, un eXit è considerato includere certe manipolazioni dei dati associate ad esso (p.es. formattazione e instradamento associati ai dati che devono essere inviati all'esterno del confine).”

Read (R)

“Un Read (R) è un movimento di dati che muove un gruppo di dati dalla memoria persistente verso il processo funzionale che lo richiede. Si noti che nel metodo COSMIC-FFP, un Read è considerato includere certe manipolazioni dei dati associate ad esso, necessarie per ottenere il Read.”

Write (W)

“Un Write (W) è un movimento di dati che muove un gruppo di dati situato all'interno di un processo funzionale verso la memoria persistente. Si noti che nel metodo COSMIC-FFP, un Write è considerato includere certe manipolazioni dei dati associate ad esso, necessarie per ottenere il Write.”

Regola – ‘De-duplicazione’ dei movimenti di dati

(Il termine ‘de-duplicazione’ significa ‘eliminazione di elementi duplicati da un elenco’.) Un movimento di dati di un certo tipo (E, X, R o W), che muove un qualsiasi gruppo di dati (cioè dati riguardanti un unico oggetto d'interesse), è in generale identificato una sola volta in un qualsiasi processo funzionale.

Regola – Movimenti di Dati di Read e di Write nei Processi di ‘Aggiornamento’

Molto comunemente nel software applicativo aziendale interattivo (‘online’), un FUR di aggiornamento di dati persistenti implica l’esistenza di due processi funzionali distinti. Il primo è un ‘reperimento pre-aggiornamento’, in cui i dati riguardanti l’oggetto (gli oggetti) d’interesse da aggiornare sono in un primo momento letti (‘Read’) e visualizzati. Questo processo è seguito da un processo funzionale di ‘aggiornamento’, in cui i dati modificati sono resi persistenti da uno o più movimenti di dati di tipo Write. Il processo funzionale di reperimento pre-aggiornamento contenente solo il/i Read e nessun Write consente all’utente mano di verificare che siano state selezionate la/le corrette occorrenze di dati (‘record’). Il successivo processo funzionale di aggiornamento consente all’utente di inserire i dati che dovrebbero essere modificati, aggiunti o cancellati e di completare l’aggiornamento tramite il/i Write. Tuttavia, esistono anche varie circostanze che possono portare al FUR di un solo processo funzionale, in cui un Read è seguito da un Write dei ‘medesimi dati’, senza intervento umano.

Per ‘medesimi dati’ si intende:

- il gruppo di dati scritto è identico al gruppo di dati letto, ma i suoi valori sono stati aggiornati, o
- il gruppo di dati scritto rappresenta il medesimo oggetto d’interesse del gruppo di dati letto, ma il gruppo di dati scritto differisce da quello letto a causa, per esempio, dell’aggiunta di attributi dei dati.

In questi casi si dovrebbero identificare un Read e un Write dei medesimi dati nell’unico processo funzionale.

La Figura 3.6 illustra le relazioni generali tra i quattro tipi di sotto-processo, il processo funzionale a cui essi appartengono e il confine del software misurato.

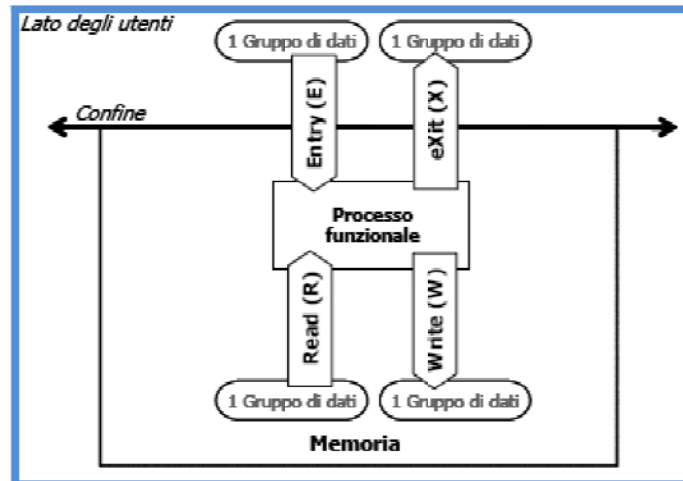


Figura 3.6 -I quattro tipi di movimento di dati e la loro relazione con il processo funzionale e i gruppi di dati.

In seguito alla descrizione della corrispondenza dei movimenti di dati attraverso i confini tra strati e tra elementi di pari livello nell'ambito di uno strato, fornita nella sezione 3.2, si hanno le seguenti regole:

Regola	Descrizione
a	Un Entry diretto a un elemento software, proveniente da un elemento di pari livello appartenente al medesimo strato, corrisponde a un eXit per l'elemento di pari livello.
b	Un eXit proveniente da un elemento software, diretto a un elemento di pari livello appartenente al medesimo strato, corrisponde a un Entry per l'elemento di pari livello.
c	Un Write è un movimento di dati verso la memoria persistente. Esso non attraversa il confine del processo funzionale a cui appartiene. Tuttavia se il Write è delegato al software di uno strato 'inferiore' distinto, ciò corrisponderà a un Entry attraverso il confine dello strato inferiore.
d	Un Read è un movimento di dati proveniente dalla memoria persistente. Esso non attraversa il confine del processo funzionale a cui appartiene. Tuttavia se il Read è delegato al software di uno strato 'inferiore' distinto, ciò corrisponderà a una coppia Entry/eXit attraverso il confine dello strato inferiore.
e	In tutti i quattro casi precedenti, si includano i movimenti di dati da ogni lato del confine nella dimensione delle loro rispettive componenti.

Movimento di dati di tipo Entry

Un Entry include tutte le manipolazioni di formattazione e presentazione dei dati richieste dagli utenti insieme con tutte le convalide associate ai dati inseriti, nella misura in cui queste manipolazioni non comportano un altro movimento di dati. Per esempio, un Entry include tutte le manipolazioni citate, eccetto il o i Read che potrebbero essere richiesti per convalidare dei codici inseriti o per ottenere delle descrizioni associate ai dati. Un Entry include anche qualsiasi funzionalità di ‘richiesta di ricezione dell’Entry (‘dei dati in entrata’).

Movimento di dati di tipo eXit

Un eXit include tutte le manipolazioni di formattazione e presentazione dei dati richieste dagli utenti, inclusa l’elaborazione richiesta per instradare l’output verso questi utenti, nella misura in cui queste manipolazioni non comportano un altro (tipo di) movimento di dati. Per esempio³³, un eXit include tutte le elaborazioni necessarie per formattare e preparare per la stampa alcuni attributi dei dati, incluse le intestazioni, o etichette, leggibili dei campi, eccetto il o i Read o Entry che potrebbero essere richiesti per fornire il valore di alcuni degli attributi dei dati stampati.

Movimento di dati di tipo Read

Un Read include tutte le elaborazioni e/o i calcoli necessari per leggere i dati, nella misura in cui queste manipolazioni non comportano un altro (tipo di) movimento di dati. Per esempio, un Read include tutti i calcoli matematici e le elaborazioni logiche necessarie per reperire un gruppo di dati o un certo numero di attributi dei dati da un gruppo di dati dalla memoria persistente, ma non la manipolazione di questi attributi dopo che il gruppo di dati è stato ottenuto. Un Read include anche qualsiasi funzionalità di ‘richiesta di accesso/lettura’.

Movimento di dati di tipo Write

Un Write include tutte le elaborazioni e/o i calcoli necessari per creare gli attributi dei dati di un gruppo di dati che deve essere scritto, nella misura in cui queste manipolazioni non comportano un altro movimento di dati. Per esempio, un Write include tutti i calcoli matematici e le elaborazioni logiche necessarie per aggiornare un gruppo di attributi dei dati che deve essere scritto eccetto qualsiasi Read o Entry che potrebbero essere richiesti per fornire il valore di altri attributi dei dati facenti parte del gruppo che deve essere scritto.

Identificazione degli Entry

Una volta identificato un possibile movimento di dati di tipo Entry, deve soddisfare i seguenti principi:

Principio	Descrizione
a	Il movimento di dati riceve attributi dei dati situati all'esterno del confine del software, dal lato dell'utente.
b	Il movimento di dati riceve attributi dei dati provenienti da un solo gruppo di dati, cioè dati riguardanti un unico oggetto d'interesse. Se si ricevono input da più di un gruppo di dati, si identifichi un Entry per ogni gruppo.
c	Il movimento di dati non invia dati all'esterno attraverso il confine, né legge o scrive dati.
d	Nell'ambito del processo funzionale in cui è identificato, il movimento di dati è unico, cioè l'elaborazione e gli attributi dei dati identificati sono differenti da quelli di altri Entry inclusi nel medesimo processo funzionale.

Le seguenti regole possono essere utili per convalidare lo stato di ogni candidato movimento di dati di tipo Entry:

Regola	Descrizione
a	Gli eventi d'innesco basati sul clock sono sempre esterni al software misurato. Quindi, un evento che avviene ogni 3 secondi è associato a un Entry che muove un attributo dei dati, per esempio. Si noti che anche nel caso in cui un tale evento d'innesco sia generato periodicamente non dall'hardware, ma da un processo funzionale software, quest'ultimo può essere ignorato dalla misurazione poiché ha luogo, per definizione, all'esterno del confine del software misurato.
b	A meno che non sia necessario a tal fine uno specifico processo funzionale, ottenere il valore di tempo dal clock del sistema non è considerato essere un Entry. Per esempio, quando un processo funzionale scrive un riferimento temporale ('time stamp'), non si identifica alcun Entry per l'ottenimento del valore del clock del sistema.

Identificazione degli eXit

Una volta identificato un possibile movimento di dati di tipo eXit, deve soddisfare i seguenti principi:

Principio	Descrizione
a	Il movimento di dati invia attributi dei dati all'esterno del confine, verso il lato dell'utente.
b	Il movimento di dati invia attributi dei dati appartenenti a un solo gruppo di dati, cioè dati riguardanti un unico oggetto d'interesse. Se si inviano all'esterno del confine del software dati appartenenti a più di un gruppo di dati, si identifichi un eXit per ogni gruppo di dati referenziato.
c	Il movimento di dati non riceve dati dall'esterno del confine, né legge o scrive dati.
d	Nell'ambito del processo funzionale in cui è identificato, il movimento di dati è unico, cioè l'elaborazione e gli attributi dei dati identificati sono differenti da quelli di altri eXit incluso nel medesimo processo funzionale.

La seguente regola può essere utile per convalidare lo stato di ogni candidato movimento di dati di tipo eXit:

Regola 'eXit'
<p>Quando si misura dal Punto di Vista della Misurazione dell'Utente Finale, per convenzione tutti i messaggi del software privi di dati dell'utente (p.es. messaggi di conferma o di errore) sono considerati essere occorrenze distinte di un solo tipo di messaggio. Quindi, si identifica un unico eXit per rappresentare tutti questi messaggi nell'ambito del processo funzionale in cui questi messaggi sono identificati.</p> <p>Esempio: si considerino i due processi funzionali 'A' e 'B' identificati nell'ambito del medesimo strato. 'A' può emettere potenzialmente 2 distinti messaggi di conferma e 5 messaggi di errore ai propri utenti e 'B' può emettere potenzialmente 8 messaggi di errore ai propri utenti. In questo esempio, si identificherebbero un eXit nell'ambito del processo funzionale 'A' (che gestisce $5 + 2 = 7$ messaggi) e un distinto eXit nell'ambito del processo funzionale 'B' (che gestisce 8 messaggi).</p>

Identificazione degli Read

Una volta identificato un possibile movimento di dati di tipo Read, deve soddisfare i seguenti principi:

Principio	Descrizione
a	Il movimento di dati reperisce attributi dei dati da un gruppo di dati nella memoria persistente.
b	Il movimento di dati reperisce attributi dei dati appartenenti a un solo gruppo di dati, cioè dati riguardanti un unico oggetto d'interesse. Si identifichi un Read per ogni oggetto d'interesse i cui attributi dei dati sono reperiti in un qualsiasi processo funzionale (si veda anche la regola di 'De-duplicazione' che può prevalere su questa regola).
c	Il movimento di dati non riceve o invia dati all'esterno attraverso il confine, né scrive dati.
d	Nell'ambito del processo funzionale in cui è identificato, il movimento di dati è unico, cioè l'elaborazione e gli attributi dei dati identificati sono differenti da quelli di qualsiasi altro Read incluso nel medesimo processo funzionale.
e	Durante un processo funzionale, il Read (o un Write) di un gruppo di dati può essere eseguito solo sui dati che descrivono un oggetto d'interesse per l'utente. Costanti o variabili interne al processo funzionale, risultati intermedi in un calcolo o dati memorizzati da un processo funzionale risultante solo dall'implementazione, piuttosto che dai FUR, non sono gruppi di dati e non sono presi in considerazione nella dimensione funzionale.

Identificazione dei Write

Una volta identificato un possibile movimento di dati di tipo Write, deve soddisfare i seguenti principi:

Principio	Descrizione
a	Il movimento di dati muove attributi dei dati verso un gruppo di dati dal lato software della memoria persistente.
b	Il movimento di dati muove valori di attributi dei dati appartenenti a un solo gruppo di dati, cioè dati riguardanti un unico oggetto d'interesse. Si identifichi un Write per ogni oggetto d'interesse i cui attributi dei dati sono referenziati in un qualsiasi processo funzionale (si veda anche la regola di 'De-duplicazione' che può prevalere su questa regola).

c	Il movimento di dati non riceve o invia all'esterno dati attraverso il confine, né legge dati.
d	Nell'ambito del processo funzionale in cui è identificato, il movimento di dati è unico, cioè l'elaborazione e gli attributi dei dati identificati sono differenti da quelli di qualsiasi altro Write incluso nel medesimo processo funzionale.
e	Il requisito di cancellazione di un gruppo di dati dalla memoria persistente è misurato come un singolo movimento di dati di tipo Write.
f	Durante un processo funzionale, il passo di memorizzazione di un gruppo di dati che non persiste quando il processo funzionale è completato non è un Write; esempi sono l'aggiornamento di variabili interne al processo funzionale o la produzione di risultati intermedi in un calcolo.

Estensioni locali

Il metodo di misurazione COSMIC-FFP non è attualmente progettato per fornire una modalità standard che tenga conto della dimensione di certi tipi di FUR, in particolare quelli riguardanti algoritmi matematici complessi o sequenze complesse di regole, quali quelle contenute nei sistemi esperti. Tuttavia, si può avere che, nel contesto locale di un'organizzazione che usa il metodo di misurazione COSMIC-FFP, sia possibile tener conto di queste funzionalità in una modalità significativa come standard locale.

Per questa ragione, il metodo di misurazione COSMIC-FFP prevede la possibilità di estensioni locali. Dato un qualsiasi processo funzionale contenente un sotto-processo funzionale di manipolazione dei dati eccezionalmente complesso, il misuratore è libero di stabilire le proprie unità localmente determinate di dimensione funzionale per queste funzionalità di carattere eccezionale.

3.2.4.2 Applicazione della funzione di misurazione

Questo passo consiste nell'applicare la funzione di misurazione COSMIC-FFP a ognuno dei movimenti di dati identificati in ogni processo funzionale.

Funzione di misurazione COSMIC-FFP

“La funzione di misurazione COSMIC-FFP è una funzione matematica che assegna un valore al proprio argomento sulla base dello standard di misurazione COSMIC-FFP (cfr. definizione seguente). L'argomento della funzione di misurazione COSMIC-FFP è il movimento di dati.”

Standard di misurazione COSMIC-FFP

“Lo standard di misurazione COSMIC-FFP, 1 *cfsu* (Cosmic Functional Size Unit – Unità di Dimensione Funzionale COSMIC), è definito come la dimensione di un movimento di dati elementare.”

Secondo questa funzione di misurazione, a ogni istanza di movimento di dati (Entry, eXit, Read o Write) identificato nel passo 1 (Figura 3.5) si assegna una dimensione numerica di 1 *cfsu*.

3.2.4.3 Aggregazione dei risultati della funzione di misurazione

Questo passo consiste nell’aggregare i risultati della funzione di misurazione, applicata a tutti i movimenti di dati identificati, in un unico valore di dimensione funzionale. Questo passo è svolto attenendosi ai seguenti principi.

Principio	Descrizione
a	Per ogni processo funzionale, le dimensioni funzionali dei suoi singoli movimenti di dati sono aggregate in un unico valore di dimensione funzionale sommandole aritmeticamente tra di loro: $\text{dim.cfsu (processo funz.le i)} = \sum \text{dim. (Entryi)} + \sum \text{dim. (eXiti)} + \sum \text{dim. (Readi)} + \sum \text{dim. (Writei)}.$
b	Per ogni processo funzionale, la dimensione funzionale delle modifiche apportate o da apportare ai FUR è aggregata a partire dalle dimensioni dei corrispondenti movimenti di dati modificati, secondo la seguente formula: $\text{dim.cfsu (modifica (processo funz.le i))} = \sum \text{dim. (dati aggiunti)} + \sum \text{dim. (dati modificati)} + \sum \text{dim. (dati cancellati)}.$
c	La dimensione di ogni porzione del software, appartenente a un dato strato, si ottiene aggregando la dimensione dei nuovi processi funzionali e di ogni processo funzionale modificato nell’ambito dei FUR individuati per ogni porzione.
d	Le dimensioni di strati distinti o di porzioni di software distinte nell’ambito degli strati possono essere sommate tra di loro solo se misurate dal medesimo Punto di Vista della Misurazione.
e	Inoltre, le dimensioni di porzioni di software appartenenti a un qualsiasi strato o a strati differenti possono essere sommate tra di loro solo se ha senso farlo, alla luce dello Scopo della Misurazione. (Per esempio, se varie componenti principali sono sviluppate con differenti tecnologie, da differenti sotto-gruppi di progetto, allora può non esserci alcun valore pratico nel sommare tra di loro le dimensioni di tali componenti.)

3.2.5 Registrazione delle misurazioni COSMIC-FFP

Vi sono delle convenzioni da seguire nella stesura, nella registrazione e nella archiviazione delle misurazioni COSMIC-FFP. Qui di seguito verranno mostrate quali sono queste regole.

3.2.5.1 Etichettatura dei risultati di misurazione COSMIC-FFP

Quando si registra una dimensione funzionale COSMIC-FFP, la si dovrebbe etichettare attenendosi alla seguente convenzione, in conformità allo standard ISO/IEC 14143-1:1998.

Regola 'Etichettatura di una misurazione COSMIC-FFP'

Un risultato di misurazione COSMIC-FFP è riportato come:

"**x** cfsu (v. **y**)", dove

- '**x**' rappresenta il valore numerico della dimensione funzionale in unità di dimensione funzionale COSMIC (CFSU – COSMIC Functional Size Units),
- '**v. y**' rappresenta l'identificativo della versione standard del metodo COSMIC-FFP usata per ottenere il valore numerico della dimensione funzionale '**x**'.

Esempio: un risultato ottenuto mediante le regole dell'attuale versione 2.2 del Manuale di Misurazione COSMIC-FFP è riportato come "x cfsu (COSMIC-FFP v. 2.2)".

NOTA Se si è fatto uso di un metodo di approssimazione locale per ottenere la misurazione, ma, a parte ciò, la misurazione è stata effettuata tramite le convenzioni di una versione standard del metodo COSMIC-FFP, si usi la precedente convenzione di etichettatura, ma si annoti altrove l'utilizzo del metodo di approssimazione.

Qualora si sia fatto uso di estensioni locali, come definito nella precedente sezione 3.2.4.1 (pag. 62), si deve registrare il risultato della misurazione come definito di seguito.

Regola 'Etichettatura delle estensioni locali COSMIC-FFP'

Un risultato di misurazione COSMIC-FFP ottenuto mediante estensioni locali è riportato come:

“ x cfsu (v. y) + z fsu locali“, dove

- ' x ' rappresenta il valore numerico ottenuto aggregando tutti i singoli risultati di misurazione conformemente al metodo COSMIC-FFP standard, in unità di dimensione funzionale COSMIC (CFSU – COSMIC Functional Size Units),
- ' $v. y$ ' rappresenta l'identificativo della versione standard del metodo COSMIC-FFP usata per ottenere il valore numerico della dimensione funzionale ' x ',
- ' z ' rappresenta il valore numerico ottenuto aggregando tutti i singoli risultati di misurazione ottenuti mediante estensioni locali del metodo COSMIC-FFP, in unità di dimensione funzionale locali (Local FSU – Local Functional Size Units).

3.2.5.2 Archiviazione dei risultati di misurazione COSMIC-FFP

Quando si archiviano i risultati di misurazione COSMIC-FFP, si dovrebbero mantenere le seguenti informazioni, per garantire che il risultato sia sempre interpretabile.

Si mantiene una registrazione distinta per ogni strato, o componente appartenente a uno strato, misurati. Questa registrazione contiene almeno le seguenti informazioni:

Convenzione	'Dettaglio di misurazione COSMIC-FFP'
a	Identificativo della componente software misurata (nome, ID di versione o ID di configurazione).
b	Descrizione della funzione dello strato.
c	Scopo della Misurazione.
d	Ambito della Misurazione.
e	Punto di Vista della Misurazione.
f	Momento del ciclo di vita del progetto in cui si è svolta la misurazione (specialmente qualora la misurazione sia una stima basata su FUR incompleti, o sia stata effettuata sulla base di funzionalità effettivamente consegnate).
g	Accuratezza, ricercata o valutata, della misurazione.

h	Se si è applicato il metodo di misurazione COSMIC-FFP standard e/o un'approssimazione locale del metodo standard e/o estensioni locali (cfr. sezione 3.2.4.1). Si usino le convenzioni di etichettatura delle sezioni 3.2.5.1 o 3.2.5.2.
i	Se la misurazione è di funzionalità sviluppate o consegnate (le funzionalità 'sviluppate' sono ottenute creando nuovo software; le funzionalità 'consegnate' includono le funzionalità 'sviluppate' e anche le funzionalità ottenute con altri mezzi rispetto alla creazione di nuovo software, cioè incluse tutte le forme di riuso di software esistente, l'uso di parametri esistenti per aggiungere o modificare funzionalità, ecc.).
j	Se la misurazione è di funzionalità fornite ex novo o è il risultato di un'attività 'evolutiva' (la somma cioè delle funzionalità aggiunte, modificate e cancellate).
k	Descrizione dell'architettura degli strati in cui si effettua la misurazione, se applicabile, percepita dal Punto di Vista della Misurazione.
l	Numero di componenti principali, se applicabile, le cui dimensioni siano state sommate tra di loro per dare la dimensione totale registrata.
m	Inventario dei processi funzionali identificati, includente per ogni processo funzionale: <ul style="list-style-type: none">• nome,• numero di movimenti di dati di tipo Entry,• numero di movimenti di dati di tipo eXit,• numero di movimenti di dati di tipo Read,• numero di movimenti di dati di tipo Write.

4. QUALCHE PASSO INDIETRO: LE LOC

4.1 Definizione

“Le LOC sono una metrica del software utilizzata per misurare la grandezza di un programma tramite il conteggio del numero di linee nel testo dei codici sorgente dei programmi. E’ un metodo tipicamente usato per predire i costi che è necessario affrontare per sviluppare un software.”

Il metodo delle LOC ricalca quindi gli obiettivi dei Function Point basando le sue stime sulle linee di codice di un programma.

4.2 Conteggio LOC

Uno dei principali inconvenienti nell’utilizzo di questo metodo è la sua scarsa duttilità ed usabilità in situazioni differenti: è infatti molto difficile utilizzare questo metodo per confrontare software con un numero di linee di codice diverso tra loro. Facciamo un paragone con i Function Point: le LOC sono limitate al conteggio delle linee di codice e su esse si basa il loro confronto mentre i Function Point si basano su tutti gli aspetti che riguardano il software permettendo così una valutazione accurata che va oltre il semplice conteggio e consente di confrontare programmi che possono avere anche differenze enormi dal punto di vista del numero di linee, cosa che non è possibile fare con le LOC.

La misura delle LOC si divide in due tipi: LOC Fisiche e LOC Logiche.

Ciò che differenzia i due tipi di conteggio è che le LOC Fisiche inseriscono nei propri calcoli anche le linee di commento. Vengono inoltre contate le linee bianche fino a che queste non arrivino al 25% del totale di linee calcolate; una volta raggiunta questa soglia si smette di calcolarle.

Le LOC Logiche invece non tengono conto della formattazione del codice e delle convenzioni di stile adottate tipicamente nella stesura del codice.

Questa differenza risulta essere rilevante ma molto spesso, si dimentica di specificare che tipo di misurazione è stata fatta, rendendo poco chiaro il lavoro svolto.

Un aspetto controverso è legato all’utilizzo di questo metodo di misurazione e i programmi che va a valutare: il metodo è sì utile per misurare con efficienza il costo per la realizzazione di un software ma non tiene conto di una variabile importante: l’abilità dello sviluppatore.

Vediamo un esempio:

Due programmatori, A e B, decidono di sviluppare lo stesso programma. Il programmatore A ha un'esperienza maggiore mentre il programmatore B è un ragazzo alle prime armi.

Il programmatore non ha il semplice ruolo di scrivere il codice ma anche di ragionare su di esso e migliorarlo; per questo il programmatore A è in grado di scrivere con meno righe di codice un programma efficiente e del tutto simile a quello scritto dal programmatore B, utilizzando quindi meno righe di codice. Facendo un confronto tra i due programmi utilizzando il metodo delle LOC si avrebbe come risultato che il programma con un numero di linee maggiore è più complesso e completo di quello con meno righe di codice quando invece sappiamo che lo studio che è stato fatto dal programmatore A per migliorare il livello di astrazione e semplificare il codice rende il suo programma più completo.

Un altro fattore di cui bisogna tener conto è il tipo di linguaggio utilizzato. Il confronto è semplice se il linguaggio utilizzato è il medesimo ma se andiamo a cambiare quello di uno dei due programmatori, le cose cambiano.

C	COBOL
<pre>#include <stdio.h> Int main (void) { Printf ("Hello Word"); return 0; }</pre>	<pre>000100 IDENTIFICATION DIVISION. 000200 PROGRAM-ID. HELLOWORLD. 000300 000400* 000500 ENVIRONMENT DIVISION. 000600 CONFIGURATION SECTION. 000700 SOURCE-COMPUTER. RM-COBOL. 000800 OBJECT-COMPUTER. RM-COBOL. 000900 001000 DATA DIVISION. 001100 FILE SECTION. 001200 100000 PROCEDURE DIVISION. 100100 100200 MAIN-LOGIC SECTION. 100300 BEGIN. 100400 DISPLAY " " LINE 1 POSITION 1 ERASE EOS. 100500 DISPLAY "Hello world!" LINE 15 POSITION 10. 100600 STOP RUN. 100700 MAIN-LOGIC-EXIT. 100800 EXIT.</pre>
Linee di Codice: 5	Linee di Codice: 17

Come si può notare in questo esempio, cambiando il linguaggio di programmazione il conteggio delle LOC cambia e non di poco, per questo il metodo è da ritenere inefficace quando si vanno a confrontare programmi sviluppati in linguaggi diversi.

5. ESEMPIO PRATICO: IL GIOCO DEGLI SCACCHI

In questa ultima sezione vedremo un esempio pratico, utile a far capire il funzionamento dei due metodi principali qui spiegati.

5.1 Diagrammi UML

Il nostro gioco è sviluppato per poter essere giocato tra un giocatore e il computer, oppure tra due giocatori umani ma non è realizzato per essere sfruttato in rete.

5.1.1 Struttura

In figura il diagramma UML che rappresenta il nostro gioco degli scacchi:

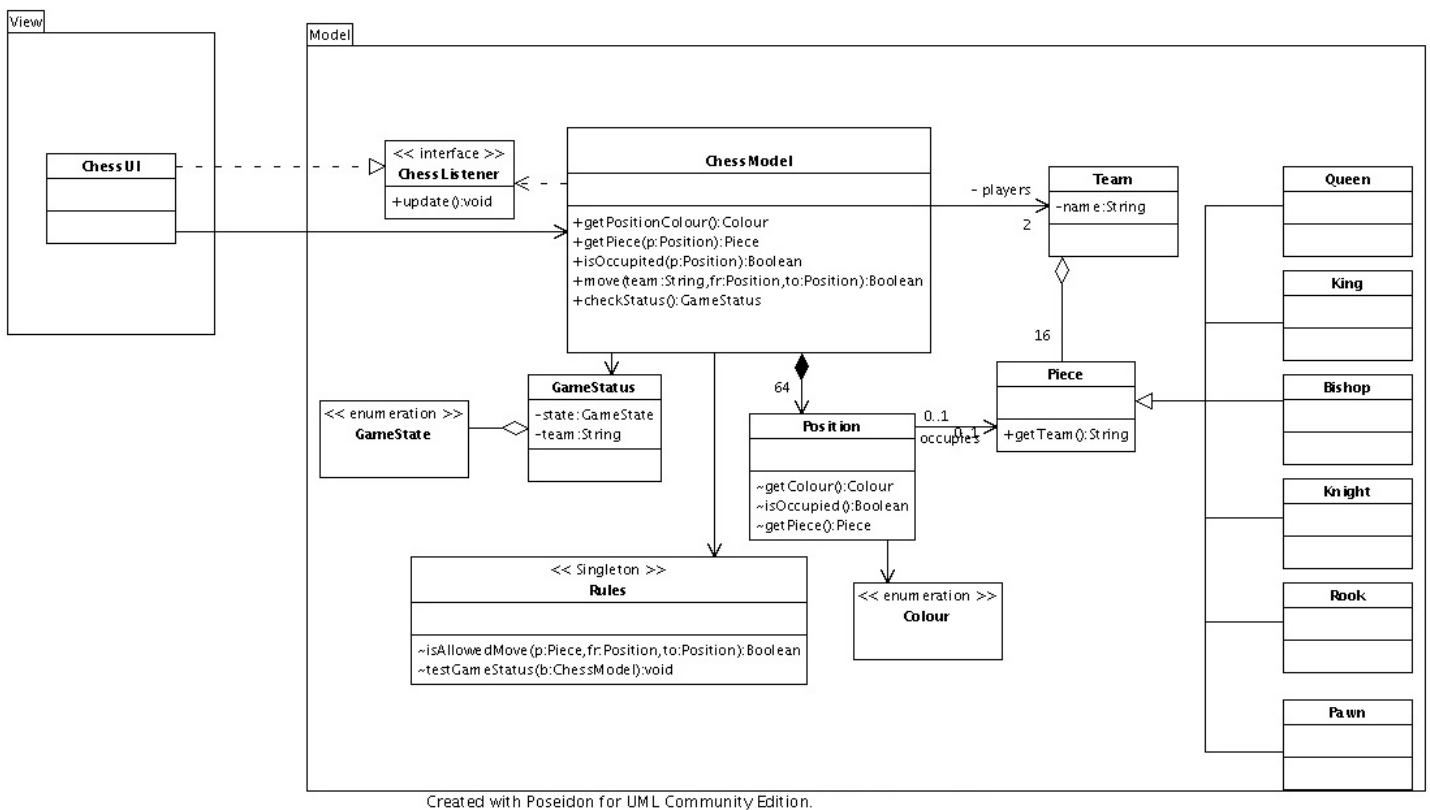


Figura 5.1 – Diagramma UML del gioco di scacchi

La classe principale è `ChessModel` e si occupa di tutta la gestione del gioco: accetta le mosse dai giocatori e se sono corrette aggiorna la scacchiera tramite un listener ad essa associato. I metodi principali di questa classe sono:

- `move (team:String, fr:Position, to:Position):Boolean.`

Funzione invocata dall'interfaccia grafica, quando il giocatore genera una mossa.

- `isOccupied (p:Position):Boolean.`

Controlla se la casella da cui si sta muovendo il pezzo è occupata utilizzando un oggetto `Position`.

- `checkStatus ():GameState.`

Controlla lo stato del gioco.

- `getPiece (p:Position):Piece.`

Ottiene il tipo di pezzo che occupa una certa posizione `p`. Anche questo è un metodo che utilizza un oggetto `Position`.

- `getPositionColour ():Colour.`

Chiede il colore di una certa posizione.

Il controllo di validità delle mosse non risiede nella classe principale, ma in una dedicata: `Rules`.

Il metodo principale di questa classe è appunto:

- `isAllowedMove (p:Piece, fr:Position, to:Position):Boolean.`

Funzione che controlla la validità della mossa dalla casella `fr` alla casella `to` del pezzo `p`.

La scacchiera è memorizzata nelle istanze delle classi `Position`, che sono in tutto 64 (una per ogni casella della scacchiera) e contengono informazioni quali il colore, se sono occupate o no, e il tipo di pezzo che, nel caso, occupa la casella. La stessa classe `Position` fa riferimento alla classe `Piece` per sapere il tipo di pezzo e a chi appartiene (se al bianco o al nero).

I metodi della classe sono:

- `getColour ():Colour.`

Funzione invocata da `ChessModel`, ritorna il colore della casella della posizione su cui si è invocato il metodo.

- `isOccupied ():Boolean.`

Altra funzione invocata da ChessModel. Verifica se la casella su cui si è chiamato il metodo è occupata.

- `getPiece():Piece`.

Funzione come `getColour` e `isOccupied`, invocata da ChessModel, restituisce il pezzo residente sulla casella su il metodo viene invocato.

Vi è infine la classe `Piece` che viene estesa dai vari tipi di pezzi. Le varie istanze di questa classe rappresentano i pezzi in gioco. Il metodo di questa classe è uno:

- `getTeam():String`

Funzione invocata da `Position` che ritorna il colore del pezzo della casella da cui è chiamato.

In definitiva l'unico momento in cui avviene una comunicazione si verifica quando un giocatore genera una mossa tramite l'interfaccia grafica. La mossa viene passata a ChessModel che ne verifica la validità.

Oltre a questo, la classe `ChessModel`, nasconde completamente la struttura del programma stesso: le regole risiedono in una classe esterna, come per la scacchiera incapsulata nella classe `Position` e `Piece` aggiungendo così un buon livello di astrazione.

5.1.2 Comunicazione

Il nucleo del programma risiede nella classe `ChessModel`. E' quindi qui che i giocatori vengono istanziati grazie alla classe `Team`. Una volta fatto ciò il gioco può partire: l'utente genera una mossa che viene passata a ChessModel che procede a verificarne la validità. Una volta eseguiti tutti i controlli la scacchiera viene aggiornata.

Ora ci serviamo di un diagramma di sequenza (Figura 5.2) per analizzare la fase di passaggio della mossa, che è il momento in cui avviene una comunicazione, in questo caso fra utente, cioè giocatore umano e il programma.

La mossa parte da `ChessUI`, ovvero è l'utente a generarla tramite l'interfaccia grafica invocando il metodo `move` della classe `ChessModel`. Come si può notare quest'ultima delega ad altre classi il compito di testare la validità della mossa. Per prima cosa interroga la classe `Position` chiamando il metodo `isOccupied` per sapere se la casella da cui viene la mossa è occupata. Se così è invoca sempre un metodo di `Position` per sapere quale pezzo è presente su quella casella.

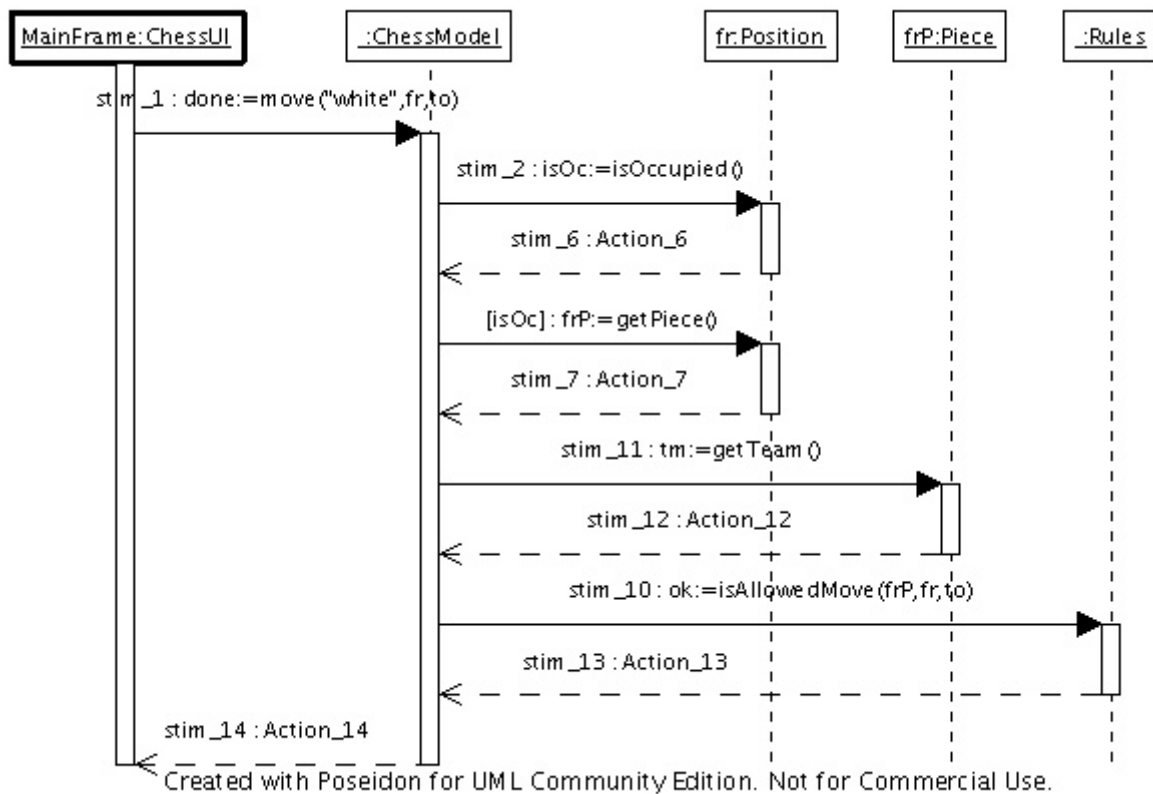


Figura 5.2 – Diagramma di sequenza UML rappresentante la sequenza di mossa

A questo punto per sapere a quale giocatore appartiene il pezzo in quella casella chiama il metodo `getTeam` della classe `Piece`. Infine si occupa della legalità della mossa andando a invocare il metodo `isAllowedMove` della classe `Rules`.

Lo stato del gioco verrà salvato e il controllo ripasserà all'interfaccia grafica per poter fare generare all'utente un'altra mossa.

La comunicazione avviene quindi solo in due momenti:

- Il giocatore comunica la sua mossa al programma principale (`ChessModel`).
- Il Programma principale, se è avvenuta una mossa valida, comunica ai giocatori in che modo aggiornare la scacchiera.

5.2 Conteggio: il metodo IFPUG 4.2.1

Questo esempio mostra una tabella riepilogativa del calcolo fatto sul gioco degli scacchi; è necessario però motivare il perché di questa valutazione spiegando anche cosa si è andato a considerare. Per rendere un po' più completa l'analisi del software si è stabilito che il gioco degli scacchi avesse funzioni standard nell'interfaccia: creazione nuova partita, salvataggio della partita in corso e consultazione di un help. Diverse azioni avranno l'opportunità di essere eseguite tramite menù a tendina oppure tramite comando rapido da tastiera; sarà inoltre disponibile una sezione in cui poter modificare le preferenze, caricarne di nuove oppure reimpostare quelle di default. Saranno infine previsti comandi come quello di abbandono della partita e di resa.

ILFs identificati:

- Chess Model
- Team
- Chess Listener
- Piece
- Rules

EIFs identificati:

- Chess UI

Le attività svolte dall'utente che sono state identificate sono:

- Creazione nuova partita
- Salvataggio partita corrente
- Carica partita preesistente
- Muovi Pezzo
- Consulta l'Help
- Controlla Validità Mossa
- Stato del gioco
- Modifica impostazioni
- Salva impostazioni
- Ripristina impostazioni iniziali
- Abbandona
- Resa

I calcoli fatti son stati eseguiti tenendo conto delle indicazioni del metodo spiegate nei paragrafi dalla pagina 12 alla 37.

5.2.1 *Gli ILFs*

Chess Model

Il numero di DET determinati per questo ILF è stato valutato nel modo seguente: secondo lo schema UML precedentemente illustrato e secondo le specifiche del metodo IFPUG, ho valutato con 5 DET le istruzioni di Chess Model, quindi un DET per ognuna di esse. Il manuale specifica inoltre che deve essere contato un DET per ogni posizione specifica ed univoca riconoscibile dall'utente; di conseguenza il sottoinsieme di Chess Model, Position, fornisce 64 posizioni univoche, riconoscibili all'utente, per le quali il conteggio complessivo è di 64 DET.

Il conteggio dei RET invece è molto semplice: si conta un RET per ogni sottogruppo dell'ILF mentre se non ce ne sono il numero di RET resta fermo a uno. Nel caso di Chess Model si ha un sottogruppo identificato in Position.

La valutazione finale di Chess Model sarà quindi Media.

Team

I DET determinati per l'ILF in questione sono 18. Questo numero è il risultato della somma dei DET ricavati dall'identificazione degli utenti, cosa che Team consente e il cui numero è due, e dal numero di DET dovuti al numero di pezzi univoci e distinguibili dall'utente che sono a disposizione del Team. L'ILF Team non ha sottogruppi pertanto il numero di RET è pari a uno per una valutazione finale Bassa.

Chess Listener

Questo è un ILF molto semplice ma allo stesso tempo importante perché fa comunicare l'interfaccia grafica con il motore del gioco Chess Model. Nonostante la sua importanza è la sua semplicità a renderlo di Bassa complessità funzionale: due soli DET ed un solo RET.

Piece

La classe Piece è molto importante perché gestisce i pezzi di ogni team (32 DET) oltre ad identificare le differenze tra i vari tipi di pezzi (6 DET). Non avendo

sottogruppi, questo ILF ha un numero di RET pari a uno. La complessità funzionale è perciò Bassa.

Rules

Rules è una classe importantissima perché stabilisce la correttezza delle mosse da parte degli utenti. Essendo creata però al solo scopo di verifica delle mosse, il numero di DET e di RET è esiguo pertanto la complessità funzionale è Bassa.

5.2.2 *Gli EIFs*

ChessUI

Essendo poco dettagliato, lo schema UML non consente di fare analisi più approfondite su questa classe che rimane pertanto a livelli di complessità Bassa.

5.2.3 *Gli EIs*

Nuova Partita

L'avvio di una nuova partita apre una maschera preliminare per determinare alcune impostazioni di partenza come ad esempio il nome del Team. Questa funzione impone inoltre la creazione della scacchiera e dei pedoni chiamando in causa la classe Chess Model. In questa operazione c'è quindi la lettura di ogni ILFs precedentemente analizzato per un totale di 4 FTR.

Avendo stabilito in fase preliminare che i giocatori potevano essere anche due, il numero di utenti identificabili è due; ad essa va aggiunta la capacità di inviare messaggi informativi da parte della funzione Nuova Partita che quindi avrà un numero di DET pari a 3.

Così come spiegato nelle tabelle del capitolo 2, 4 FTR e 3 DET determinano una complessità funzionale Media.

Muovi Pezzo

Il movimento di un pezzo comporta lavoro per il Chess Model, per Rules e Team ma anche per il Chess Listener che deve prendere in input la mossa dell'utente, passarla al Chess Model e rimandare indietro una risposta positiva o negativa. Vi è quindi il mantenimento di quattro ILF e la lettura di uno di essi per un totale di 5 FTR e 3 DET dovuti all'identificazione degli utenti e alla capacità di invio di un

messaggio di errore in caso di mossa non consentita. La valutazione sarà quindi di complessità funzionale Media.

Carica Partita

Il caricamento della partita va a mantenere il Chess Model e il Team in quanto recuperando il file del salvataggio tramite un processo di lettura (1 FTR), vengono ricaricate le posizioni dei pezzi e la scacchiera oltre che al nome del team. Al totale di 3 FTR si accompagnano 3 DET dovuti all'identificazione degli utenti e all'invio di messaggi. Il livello di complessità funzionale è quindi Medio.

Modifica Impostazioni

Il programma dà la possibilità all'utente di modificare alcune impostazioni per personalizzare il gioco: avrà quindi la possibilità di scegliere il colore dei pezzi o modificare il nome del proprio team. Il conteggio dei DET e dei RET è rispettivamente di 4 e 3 per una complessità funzionale Media.

Abbandona

Abbandona consente all'utente di terminare la partita corrente senza che essa venga dichiarata come persa. Il comando è molto semplice e coinvolge ChessModel, Chess Listener e Team. La sua complessità funzionale è Bassa.

Resa

La differenza con Abbandona è puramente a livello statistico poiché dichiarando la resa la partita sarà considerata persa. Il calcolo e la complessità funzionale sono del tutto simili ad comando Abbandona perciò Basso.

5.2.4 *Gli EOs*

Salva Partita

Il salvataggio della partita è stato identificato come EO poiché invia dati all'esterno del confine dell'applicazione. Legge Chess Model, Team, Piece e ChessUI determinando così un numero di FTR pari a 4. Il numero di DET è pari a 4: identificazione degli utenti, invio di dati al di fuori del confine e possibilità di utilizzare comandi alternativi per lo stesso processo logico sono le determinanti di questo valore. La complessità funzionale è quindi Media.

Stato del gioco

Questo comando semplicemente da un riepilogo del numero di mosse e del tempo trascorso oltre a comunicare i pezzi mangiati all'avversario. Dovendo gestire più di un utente e dovendo leggere in tutti gli ILF del programma (escluso Rules) la complessità funzionale è Media (4 DET, 4 RET).

Salva impostazioni

Così come il salvataggio della partita, il salvataggio delle impostazioni genera o aggiorna un file di config all'esterno del confine del sistema; la differenza con Salva Partita sta nel fatto che non viene letto l'ILF Piece. Il numero di DET è quindi 3 mentre il numero di RET è 4; la complessità funzionale è Bassa.

Reimposta impostazioni

Reimpostare le preferenze di default è il procedimento opposto del Salvataggio delle impostazioni. Gli ILF chiamati in causa sono i medesimi così per il numero RET. La complessità funzionale è quindi Bassa.

5.2.5 *Gli EQs*

Consulta l'Help

La consultazione dell'help è una procedura molto semplice che impiega poche risorse del programma: l'unica operazione è quella di lettura di un ILF. L'identificazione degli utenti invece comporta l'aggiunta di due DET; l'help può esser richiamato spingendo un comando rapido da tastiera. 1 FTR e 3 DET danno come risultato una complessità funzionale Bassa.

Controllo validità mossa

Il controllo della validità di una mossa legge tre ILF, Chess Listener; Chess Model e Rules. Per quanto riguarda la determinazione del numero di DET il procedimento è lo stesso: due DET per l'identificazione degli utenti ed uno per la capacità di inviare messaggi di responso. Complessità funzionale Bassa.

5.2.6 Tabelle riepilogative

Function Type	Functional Complexity	Complexity Totals	Function Type Totals
ILFs	<u>4</u> Bassa <u>1</u> Media <u> </u> Alta	X07 = X10 = X15 =	28 10
EIFs	<u>1</u> Bassa <u> </u> Media <u> </u> Alta	X05 = X07 = X10 =	5
EIs	<u>2</u> Bassa <u>4</u> Media <u> </u> Alta	X03 = X04 = X06 =	6 16
EOs	<u>2</u> Bassa <u>2</u> Media <u> </u> Alta	X04 = X05 = X07 =	8 10
EQs	<u>2</u> Bassa <u> </u> Media <u> </u> Alta	X03 = X04 = X06 =	6
TOTALE			89 FP

L'ultima parte del conteggio è relativa alle caratteristiche generali del sistema, valutate secondo le informazioni date in precedenza:

Caratteristiche Generali del Sistema	Grado Di Influenza
1. Comunicazione dati	1
2. Distribuzione dell'elaborazione	0
3. Prestazioni	1
4. Utilizzo intensivo della configurazione	0
5. Frequenza delle transazioni	0
6. Inserimento dati interattivo	0
7. Efficienza per l'utente finale	3

8. Aggiornamento interattivo	0
9. Complessità elaborativa	1
10. Riusabilità	5
11. Facilità di installazione	1
12. Facilità di gestione operativa	5
13. Molteplicità di siti	0
14. Facilità di modifica	3
Grado Totale di Influenza	20
Fattore di Aggiustamento	0.85

La formula per il calcolo del fattore di aggiustamento è: $FA = (20 * 0,01) + 0,65$
 TOTALE FUNCTION POINT = $89 * 0.85 = 75.7 \approx 76 \text{ FP}$

5.3 Conteggio: il metodo COSMIC-FFP v. 2.2

Il metodo COSMIC prevede l'identificazione preliminare di due o più strati tra loro comunicanti nei quali verranno successivamente trovati i vari processi funzionali.

I due strati identificati sono l'Interfaccia che rappresenta l'utente dello strato Chess Model che è l'oggetto di misurazione.

5.3.1 Interfaccia: processi funzionali

Nuova partita

Questo processo ha come attributo un Read, assegnatogli perché legge un comando da utente e un eXit perché invia le informazioni all'esterno del confine verso il Chess Model.

Muovi pezzo

Il movimento di un pezzo genera la scrittura di nuove informazione verso un altro gruppo dati, ecco perché spiegata l'assegnazione di un Write.

Salva partita

Il salvataggio di una partita prende in input il comando (Read) e procede alla scrittura su un file esterno, dei dati (Write).

Caricamento partita

Il caricamento della partita prevede la lettura di un file (Read) e l'emissione dei dati ottenuti da esso (eXit).

Consulta l'help

L'unica funzione dell'help è quella di mostrare a video una pagina testuale con le spiegazioni più utili per l'utente, quindi è sufficiente un parametro eXit.

Modifica impostazioni

La modifica delle impostazioni comporta la lettura (Read) di nuove informazioni immesse dall'utente.

Salva impostazioni

Il salvataggio di queste impostazioni avviene tramite la lettura di un comando (Read) e la scrittura su un file esterno di tali impostazioni (Write).

Reimposta impostazioni

Reimposta carica da un file di default le impostazioni iniziali del programma (Read), ripristinandole e le scrive (Write) nel file di configurazione affinché esse vengano interpretate dal programma.

Abbandona

Abbandona è il comando che consente all'utente di terminare la partita senza che venga mostrato a video il risultato di quest'ultima e senza che esso influisca sulle statistiche.

Resa

Resa è il comando che consente all'utente arrendersi dichiarandosi sconfitto.

5.3.2 Chess Model: processi funzionali

Disponi i pezzi

La disposizione dei pezzi è vincolata dall'inizio di una nuova partita: per questo motivo all'eXit del processo "Nuova partita" dell'Interfaccia, corrisponde l'Entry di questo processo funzionale. Una volta letta la richiesta, il processo manda in uscita (eXit) la disposizione dei pezzi e la creazione della scacchiera.

Controlla validità mossa

Questo processo funzionale legge dal processo dell'Interfaccia "Muovi pezzo" (Read) le informazioni sulla mossa e restituisce un responso (eXit).

Stato del gioco

Lo stato del gioco mostra la disposizione dei pezzi sulla scacchiera dopo il caricamento di una partita (eXit).

Termina Partita

Termina la partita è la conseguenza della scelta, da parte dell'utente, di abbandonare o di arrendersi all'avversario. Questo processo restituirà un eXit che porterà alla terminazione dei vari processi funzionali.

Messaggi

Come ultima regola nell'identificazione dei parametri funzionali è bene inserire quello dei messaggi: si conta quindi un eXit per identificare tutti quei messaggi che escono dallo strato Chess Model e si dirigono verso lo strato Interfaccia.

5.3.3 Tabella riepilogativa

STRATO	PROCESSO FUNZIONALE	Entry	eXit	Read	Write
A (Interfaccia)	Nuova partita		1	1	
	Muovi pezzo				1
	Salva partita			1	1
	Caricamento partita		1	1	
	Consulta l'help		1		
	Modifica impostazioni			1	
	Salva impostazioni			1	1
	Reimposta impostazioni			1	1
	Abbandona			1	
	Resa			1	
B (Chess Model)	Disponi i pezzi	1	1		
	Controlla validità mossa		1	1	
	Stato del gioco		1	1	
	Termina Partita		1		
	Messaggi		1		
TOTALE		1	10	8	4

TOTALE = **23 cfsu** (v. 2.2)

Come si può notare dal risultato, il numero non è espresso in Function Point bensì in **cfsu** (**COSMIC Functional Size Units**). E' necessaria quindi un'operazione di conversione per rendere confrontabili i risultati ottenuti con il metodo IFPUG e quelli ottenuti con il metodo COSMIC: la formula sottostante è stata ricavata dall'esperienza di tanti contatori che hanno studiato i rapporti tra i Function Point e i cfsu calcolati elaborando diverse possibili soluzioni di cui si riporta la più affidabile attualmente.

$$Y = -87 + 1,2X$$

Dove

Y è il numero di cfsu e

X è il numero di Function Point

Il numero di Function sarà quindi: $X = (Y + 87) / 1,2 = \mathbf{92 FP}$

6 IL CONTEGGIO DELLE SLOC

Per poter effettuare un conteggio con le LOC è però necessario avere un programma che conti per noi il numero di righe: vi sono molti di questi programmi in rete, la maggior parte dei quali shareware. Uno di questi è **Practiline Source Code Line Counter**: la cosa più importante che questi software offrono è la possibilità di contare rapidamente il numero di righe di codice, distinguendo i tipi di linguaggi usati, le linee di commento, le linee bianche e le linee miste per poter fare analisi sempre più dettagliate.

Il conteggio delle LOC risulta essere però inconfrontabile con il numero di Function Point misurati, senza un'opportuna tecnica di trasformazione: il backfiring.

6.1 Il backfiring

Questa tecnica è molto importante perché consente di trasformare il numero di linee di codice misurate in un numero di Function Point, potendo così confrontare questi risultati tra loro.

Questo approccio è stato creato da una ricerca effettuata da Capers Jones, il quale per primo fornì una tabella dettagliata dei livelli di ogni linguaggio di programmazione: più il livello del linguaggio è alto, più linee di codice sono necessarie per determinare un function point.

Nella tabella sottostante sono mostrati i fattori di conversione dei linguaggi di programmazione più usati. La formula è la seguente:

$$\text{SLOC} = \text{UFP} \times F$$

dove UFP indica il numero di function point non pesati ed F il fattore di conversione mostrato in tabella. Ai fini di un eventuale confronto è bene far notare come la formula inversa del backfiring

$$\text{UFP} = \text{SLOC} / F$$

dia come risultato un numero di Function Point non pesati (UFP e non AFP).

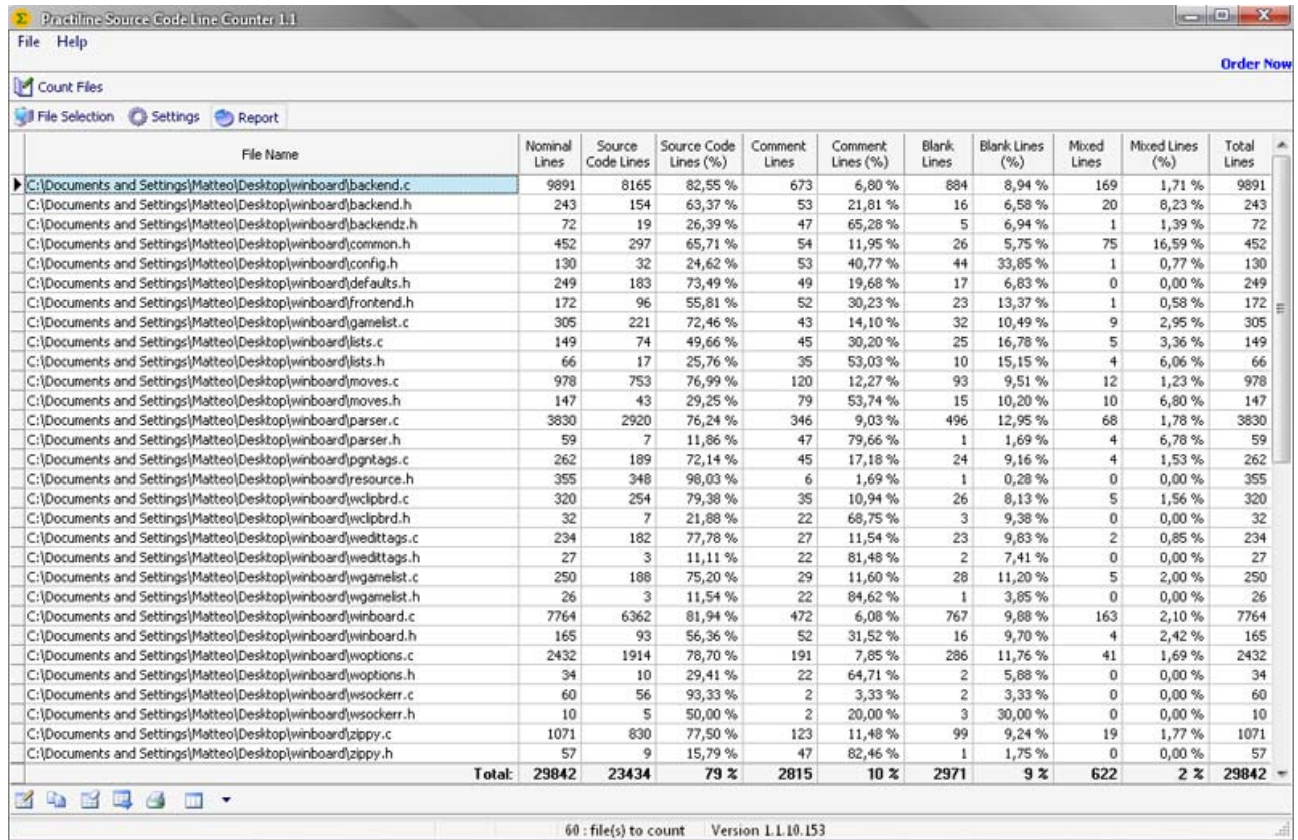
LINGUAGGIO	SLOC/UFP
Ada	71
ASP	62
APL	32
Assembly	320
Assembly (Macro)	213
ANSI/Quick/Turbo Basic	64
C	128
C++	53
ANSI Cobol 85	91
Cobol	77
Fortran 77	105
Forth	64
HTML	42
Java	59
Jovial	105
Lisp	64
Modula 2	80
Pascal	91
Prolog	64
Report Generator	80
Spreadsheet	46
SQL	35
Visual Basic	42

6.2 Conteggio con il metodo LOC: Winboard

Winboard è uno dei tanti giochi di scacchi open source reperibili in rete: si possono quindi scaricare oltre all'installante anche una cartella con il codice sorgente per poter modificare o semplicemente studiare la struttura del software.

Passiamo ora all'esempio pratico.

La figura 5.3 mostra un riepilogo dei file contenuti in Winboard e del numero di linee di codice presenti al suo interno suddivise in linee di codice, linee di commento, linee vuote e linee miste.



File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
C:\Documents and Settings\Matteo\Desktop\winboard\backend.c	9891	8165	82,55 %	673	6,80 %	884	8,94 %	169	1,71 %	9891
C:\Documents and Settings\Matteo\Desktop\winboard\backend.h	243	154	63,37 %	53	21,81 %	16	6,58 %	20	8,23 %	243
C:\Documents and Settings\Matteo\Desktop\winboard\backendz.h	72	19	26,39 %	47	65,28 %	5	6,94 %	1	1,39 %	72
C:\Documents and Settings\Matteo\Desktop\winboard\common.h	452	297	65,71 %	54	11,95 %	26	5,75 %	75	16,59 %	452
C:\Documents and Settings\Matteo\Desktop\winboard\config.h	130	32	24,62 %	53	40,77 %	44	33,85 %	1	0,77 %	130
C:\Documents and Settings\Matteo\Desktop\winboard\defaults.h	249	183	73,49 %	49	19,68 %	17	6,83 %	0	0,00 %	249
C:\Documents and Settings\Matteo\Desktop\winboard\frontend.h	172	96	55,81 %	52	30,23 %	23	13,37 %	1	0,58 %	172
C:\Documents and Settings\Matteo\Desktop\winboard\gamest.c	305	221	72,46 %	43	14,10 %	32	10,49 %	9	2,95 %	305
C:\Documents and Settings\Matteo\Desktop\winboard\lists.c	149	74	49,66 %	45	30,20 %	25	16,78 %	5	3,36 %	149
C:\Documents and Settings\Matteo\Desktop\winboard\lists.h	66	17	25,76 %	35	53,03 %	10	15,15 %	4	6,06 %	66
C:\Documents and Settings\Matteo\Desktop\winboard\moves.c	978	753	76,99 %	120	12,27 %	93	9,51 %	12	1,23 %	978
C:\Documents and Settings\Matteo\Desktop\winboard\moves.h	147	43	29,25 %	79	53,74 %	15	10,20 %	10	6,80 %	147
C:\Documents and Settings\Matteo\Desktop\winboard\parser.c	3830	2920	76,24 %	346	9,03 %	496	12,95 %	68	1,78 %	3830
C:\Documents and Settings\Matteo\Desktop\winboard\parser.h	59	7	11,86 %	47	79,66 %	1	1,69 %	4	6,78 %	59
C:\Documents and Settings\Matteo\Desktop\winboard\pgntags.c	262	189	72,14 %	45	17,18 %	24	9,16 %	4	1,53 %	262
C:\Documents and Settings\Matteo\Desktop\winboard\resource.h	355	348	98,03 %	6	1,69 %	1	0,28 %	0	0,00 %	355
C:\Documents and Settings\Matteo\Desktop\winboard\wclpbnd.c	320	254	79,38 %	35	10,94 %	26	8,13 %	5	1,56 %	320
C:\Documents and Settings\Matteo\Desktop\winboard\wclpbnd.h	32	7	21,88 %	22	68,75 %	3	9,38 %	0	0,00 %	32
C:\Documents and Settings\Matteo\Desktop\winboard\wedtags.c	234	182	77,78 %	27	11,54 %	23	9,83 %	2	0,85 %	234
C:\Documents and Settings\Matteo\Desktop\winboard\wedtags.h	27	3	11,11 %	22	81,48 %	2	7,41 %	0	0,00 %	27
C:\Documents and Settings\Matteo\Desktop\winboard\wgamelst.c	250	188	75,20 %	29	11,60 %	28	11,20 %	5	2,00 %	250
C:\Documents and Settings\Matteo\Desktop\winboard\wgamelst.h	26	3	11,54 %	22	84,62 %	1	3,85 %	0	0,00 %	26
C:\Documents and Settings\Matteo\Desktop\winboard\winboard.c	7764	6362	81,94 %	472	6,08 %	767	9,88 %	163	2,10 %	7764
C:\Documents and Settings\Matteo\Desktop\winboard\winboard.h	165	93	56,36 %	52	31,52 %	16	9,70 %	4	2,42 %	165
C:\Documents and Settings\Matteo\Desktop\winboard\woptions.c	2432	1914	78,70 %	191	7,85 %	286	11,76 %	41	1,69 %	2432
C:\Documents and Settings\Matteo\Desktop\winboard\woptions.h	34	10	29,41 %	22	64,71 %	2	5,88 %	0	0,00 %	34
C:\Documents and Settings\Matteo\Desktop\winboard\wsocerr.c	60	56	93,33 %	2	3,33 %	2	3,33 %	0	0,00 %	60
C:\Documents and Settings\Matteo\Desktop\winboard\wsocerr.h	10	5	50,00 %	2	20,00 %	3	30,00 %	0	0,00 %	10
C:\Documents and Settings\Matteo\Desktop\winboard\zippy.c	1071	830	77,50 %	123	11,48 %	99	9,24 %	19	1,77 %	1071
C:\Documents and Settings\Matteo\Desktop\winboard\zippy.h	57	9	15,79 %	47	82,46 %	1	1,75 %	0	0,00 %	57
Total:	29842	23434	79 %	2815	10 %	2971	9 %	622	2 %	29842

Figura 5.3 Conteggio LOC di Winboard con Practiline Source Code Line Counter

Come si può notare dalla lista qui sopra, Winboard è interamente sviluppato in linguaggio C, per cui il calcolo dei function point sarà il seguente:

$$\text{UFP} = 23434 / 128 \approx \mathbf{183 \text{ FP}}$$

6.3 Conteggio con il metodo LOC: Slow Chess

Slow Chess è un programma open source di scacchi interamente sviluppato in C++. Ad un primo sguardo della Figura 5.4 si può notare come il numero di linee di codice rispetto a Winboard sia nettamente inferiore. Vediamo però il numero di Function Point.

File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
C:\Documents and Settings\Matteo\Desktop\Slow296src\bibbases.cpp	269	219	81,41 %	26	9,67 %	22	8,18 %	2	0,74 %	269
C:\Documents and Settings\Matteo\Desktop\Slow296src\board.cpp	448	359	80,13 %	36	8,04 %	48	10,71 %	5	1,12 %	448
C:\Documents and Settings\Matteo\Desktop\Slow296src\data.h	644	555	86,18 %	22	3,42 %	65	10,09 %	2	0,31 %	644
C:\Documents and Settings\Matteo\Desktop\Slow296src\EndgameEval.cpp	509	389	76,42 %	45	8,84 %	47	9,23 %	28	5,50 %	509
C:\Documents and Settings\Matteo\Desktop\Slow296src\evaluate.h	91	64	70,33 %	8	8,79 %	15	16,48 %	4	4,40 %	91
C:\Documents and Settings\Matteo\Desktop\Slow296src\Evaluate.cpp	920	698	75,87 %	100	10,87 %	108	11,74 %	14	1,52 %	920
C:\Documents and Settings\Matteo\Desktop\Slow296src\gui.cpp	497	364	73,24 %	56	11,27 %	68	13,68 %	9	1,81 %	497
C:\Documents and Settings\Matteo\Desktop\Slow296src\gui.h	42	35	83,33 %	1	2,38 %	6	14,29 %	0	0,00 %	42
C:\Documents and Settings\Matteo\Desktop\Slow296src\HashNC.cpp	356	236	66,29 %	72	20,22 %	44	12,36 %	4	1,12 %	356
C:\Documents and Settings\Matteo\Desktop\Slow296src\movegenNC.cpp	502	367	73,11 %	69	13,75 %	66	13,15 %	0	0,00 %	502
C:\Documents and Settings\Matteo\Desktop\Slow296src\moves.cpp	1067	773	72,45 %	128	12,00 %	129	12,09 %	37	3,47 %	1067
C:\Documents and Settings\Matteo\Desktop\Slow296src\moves.h	51	32	62,75 %	4	7,84 %	15	29,41 %	0	0,00 %	51
C:\Documents and Settings\Matteo\Desktop\Slow296src\oBook.cpp	307	223	72,64 %	41	13,36 %	38	12,38 %	5	1,63 %	307
C:\Documents and Settings\Matteo\Desktop\Slow296src\oBook.h	77	63	81,82 %	7	9,09 %	7	9,09 %	0	0,00 %	77
C:\Documents and Settings\Matteo\Desktop\Slow296src\search.h	71	55	77,46 %	6	8,45 %	9	12,68 %	1	1,41 %	71
C:\Documents and Settings\Matteo\Desktop\Slow296src\searchExtra.cpp	620	470	75,81 %	75	12,10 %	66	10,65 %	9	1,45 %	620
C:\Documents and Settings\Matteo\Desktop\Slow296src\searchMainNC.cpp	805	560	69,57 %	107	13,29 %	107	13,29 %	31	3,85 %	805
C:\Documents and Settings\Matteo\Desktop\Slow296src\searchTT.cpp	343	226	65,89 %	69	20,12 %	44	12,83 %	4	1,17 %	343
C:\Documents and Settings\Matteo\Desktop\Slow296src\transcript.cpp	610	459	75,25 %	68	11,15 %	78	12,79 %	5	0,82 %	610
C:\Documents and Settings\Matteo\Desktop\Slow296src\transcript.h	58	42	72,41 %	6	10,34 %	9	15,52 %	1	1,72 %	58
C:\Documents and Settings\Matteo\Desktop\Slow296src\winboard.cpp	629	511	81,24 %	49	7,79 %	68	10,81 %	1	0,16 %	629
Total:	8916	6700	75 %	995	11 %	1059	12 %	162	2 %	8916

Figura 5.4 Conteggio LOC di Slow Chess con Practiline Source Code Line Counter

$$UFP = 6700 / 53 \approx 126 FP$$

6.4 Conteggio con il metodo LOC: Migoya Chess

Migoya Chess è un semplice gioco di scacchi sviluppato da uno spagnolo, in linguaggio Visual Basic. Ecco il conteggio delle LOC e dei Function Point relativi.

File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\busqueda.bas	372	247	66,40 %	39	10,48 %	86	23,12 %	0	0,00 %	372
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\Central.bas	420	305	72,62 %	48	11,43 %	67	15,95 %	0	0,00 %	420
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\Gen_mov2.bas	140	123	87,86 %	6	4,29 %	11	7,86 %	0	0,00 %	140
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\Gen_Mov.bas	576	490	85,07 %	51	8,85 %	35	6,08 %	0	0,00 %	576
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\Inicializacion.bas	202	88	43,56 %	25	12,38 %	31	15,35 %	58	28,71 %	202
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\Readme.txt	21	14	66,67 %	0	0,00 %	7	33,33 %	0	0,00 %	21
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\TABLERO.log	5	5	100,00 %	0	0,00 %	0	0,00 %	0	0,00 %	5
C:\Documents and Settings\Matteo\Desktop\Migoya Chess\tablero.frm	1710	1452	84,91 %	42	2,46 %	41	2,40 %	175	10,23 %	1710
Total:	3446	2724	78 %	211	7 %	278	8 %	233	7 %	3446

Figura 5.4 Conteggio LOC di Slow Chess con Practiline Source Code Line Counter

$$UFP = 2724 / 42 \approx 65 \text{ FP}$$

6.5 Conteggio con il metodo LOC: Java Chess

Java Chess, come dice il nome, è sviluppato interamente in java Vediamo il conteggio dei Function Point per questa applicazione.

File Name	Nominal Lines	Source Code Lines	Source Code Lines (%)	Comment Lines	Comment Lines (%)	Blank Lines	Blank Lines (%)	Mixed Lines	Mixed Lines (%)	Total Lines
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\Game	472	366	77,54 %	23	4,87 %	75	15,89 %	8	1,69 %	472
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\Game	63	45	71,43 %	2	3,17 %	16	25,40 %	0	0,00 %	63
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\JavaC	451	332	73,61 %	20	4,43 %	91	20,18 %	8	1,77 %	451
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	63	44	69,84 %	3	4,76 %	16	25,40 %	0	0,00 %	63
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	55	35	63,64 %	4	7,27 %	16	29,09 %	0	0,00 %	55
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	65	47	72,31 %	3	4,62 %	15	23,08 %	0	0,00 %	65
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	78	54	69,23 %	4	5,13 %	20	25,64 %	0	0,00 %	78
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	57	37	64,91 %	3	5,26 %	17	29,82 %	0	0,00 %	57
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	92	69	75,00 %	4	4,35 %	19	20,65 %	0	0,00 %	92
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\actor	92	64	69,57 %	6	6,52 %	22	23,91 %	0	0,00 %	92
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\board	101	83	82,18 %	2	1,98 %	16	15,84 %	0	0,00 %	101
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\board	263	205	77,95 %	15	5,70 %	40	15,21 %	3	1,14 %	263
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\board	85	70	82,35 %	0	0,00 %	15	17,65 %	0	0,00 %	85
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\dialog	221	168	76,02 %	17	7,69 %	36	16,29 %	0	0,00 %	221
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\dialog	242	195	80,58 %	9	3,72 %	37	15,29 %	1	0,41 %	242
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\dialog	138	118	85,51 %	0	0,00 %	20	14,49 %	0	0,00 %	138
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\dialog	363	315	86,78 %	10	2,75 %	37	10,19 %	1	0,28 %	363
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\dialog	187	131	70,05 %	13	6,95 %	40	21,39 %	3	1,60 %	187
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	113	88	77,88 %	2	1,77 %	23	20,35 %	0	0,00 %	113
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	548	396	72,26 %	47	8,58 %	69	12,59 %	36	6,57 %	548
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	82	61	74,39 %	4	4,88 %	17	20,73 %	0	0,00 %	82
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	131	109	83,21 %	0	0,00 %	22	16,79 %	0	0,00 %	131
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	1220	938	76,89 %	68	5,57 %	170	13,93 %	44	3,61 %	1220
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	81	64	79,01 %	1	1,23 %	16	19,75 %	0	0,00 %	81
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	70	55	78,57 %	1	1,43 %	14	20,00 %	0	0,00 %	70
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	156	126	80,77 %	4	2,56 %	26	16,67 %	0	0,00 %	156
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	174	117	67,24 %	17	9,77 %	35	20,11 %	5	2,87 %	174
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	127	86	67,72 %	8	6,30 %	31	24,41 %	2	1,57 %	127
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	133	97	72,93 %	7	5,26 %	27	20,30 %	2	1,50 %	133
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	66	50	75,76 %	1	1,52 %	15	22,73 %	0	0,00 %	66
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	209	153	73,21 %	16	7,66 %	38	18,18 %	2	0,96 %	209
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	219	147	67,12 %	19	8,68 %	46	21,00 %	7	3,20 %	219
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	48	36	75,00 %	1	2,08 %	11	22,92 %	0	0,00 %	48
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	97	74	76,29 %	3	3,09 %	20	20,62 %	0	0,00 %	97
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	1026	798	77,78 %	71	6,92 %	147	14,33 %	10	0,97 %	1026
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	136	82	60,29 %	16	11,76 %	38	27,94 %	0	0,00 %	136
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	152	102	67,11 %	12	7,89 %	38	25,00 %	0	0,00 %	152
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	138	89	64,49 %	11	7,97 %	38	27,54 %	0	0,00 %	138
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	63	43	68,25 %	5	7,94 %	15	23,81 %	0	0,00 %	63
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	147	89	60,54 %	14	9,52 %	41	27,89 %	3	2,04 %	147
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	130	85	65,38 %	12	9,23 %	33	25,38 %	0	0,00 %	130
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	137	88	64,23 %	15	10,95 %	34	24,82 %	0	0,00 %	137
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	145	97	66,90 %	13	8,97 %	35	24,14 %	0	0,00 %	145
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	154	100	64,94 %	16	10,39 %	38	24,68 %	0	0,00 %	154
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	151	99	65,56 %	15	9,93 %	37	24,50 %	0	0,00 %	151
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	149	98	65,77 %	15	10,07 %	36	24,16 %	0	0,00 %	149
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	159	114	71,70 %	11	6,92 %	34	21,38 %	0	0,00 %	159
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\engine	146	101	69,18 %	11	7,53 %	34	23,29 %	0	0,00 %	146
C:\Documents and Settings\Matteo\Desktop\Java-Chess-0.1.0pre-alpha3\src\de\java_chess\javaChess\game'	74	58	78,38 %	1	1,35 %	15	20,27 %	0	0,00 %	74
Total	17229	13053	76 %	884	5 %	3109	18 %	183	1 %	17229

Figura 5.6 Conteggio LOC di Java Chess con Practiline Source Code Line Counter

$$UFP = 13053 / 59 \approx 221 \text{ FP}$$

6.6 Confronti

6.6.1 Confronto tra programmi reali

La tabella sottostante ci mostra un riepilogo dei conteggi appena visti, dei programmi di scacchi presi in esame. La conversione da SLOC a Function Point permette di confrontare i programmi di scacchi tra di loro: come possiamo vedere, il numero di Function Point non è dipendente dal numero di righe del programma perché se i programmi sono sviluppati in linguaggi diversi il discorso cambia. Java Chess ha un numero di righe inferiore a Winboard, circa 10000 in meno, però il numero di Function Point è superiore di quasi 100 e questo è dovuto al fatto che Java ha un fattore di conversione più basso di C.

	Winboard	Slow Chess	Migoya Chess	Java Chess
Linguaggio	C	C++	Visual Basic	Java
SLOC	23434	6700	2724	13053
Unadjusted Function Point	128	126	65	221

Possiamo quindi dire che l'evoluzione del linguaggio di programmazione è determinante in questo tipo di calcoli; prendiamo ad esempio a confronto Winboard e Slow Chess: il primo è sviluppato in C mentre il secondo in C++.

C++ è da considerare un raffinamento di C, una sua versione migliorata e la tabella di riepilogo ci mostra questa sua evoluzione: come si può notare Slow Chess ha circa un quarto del numero di righe di Winboard eppure la differenza nel numero di Function Point è di solo 2. Questo è una ulteriore dimostrazione dell'importanza del linguaggio scelto.

6.6.2 Confronto tra programmi reali e specifiche

In questo paragrafo mettiamo a confronto quanto calcolato tramite specifiche di progetto e ciò che è stato misurato su programmi reali cercando di capire quali sono i motivi che hanno portato a sostanziali differenze di misurazione.

Differenza tra i risultati ottenuti

	Winboard	Slow Chess	Migoya Chess	Java Chess	IFPUG	COSMIC-FFP
Linguaggio	C	C++	Visual Basic	Java	-----	-----
SLOC	23434	6700	2724	13053	-----	-----
UFP	128	126	65	221		
UFP 1° analisi					52	84
UFP 2° analisi					89	92

Nella tabella soprastante sono stati inseriti, oltre ai programmi reali, anche i risultati dei conteggi effettuati, tramite specifiche di progetto, con i due metodi spiegati in precedenza.

Grazie alla conversione effettuata con la formula riportata a pag. 87, il numero di cfsu è stato trasformato in Function Point rendendo così possibile fare un confronto con tutte le altre misurazioni effettuate. E' altresì chiaro che la formula utilizzata per la conversione è affidabile poiché il risultato ottenuto si discosta di solo 4 FP dal risultato ottenuto con il metodo IFPUG.

Motivi che spiegano le differenze di risultati

Confrontando invece il risultato del metodo IFPUG con ciò che è stato valutato tramite backfiring, notiamo come ci sia un margine di errore consistente, ma non improbabile da ottenere.

In una prima analisi delle specifiche si erano ottenuti risultati con una discrepanza troppo elevata rispetto ai programmi reali: questo era dovuto al fatto che durante l'analisi non erano state prese in considerazione diverse funzioni che dovevano essere invece considerate. Questo spiega la differenza di calcolo tra le due analisi.

La differenza che c'è con i programmi reali è dovuta proprio al fatto che questi ultimi hanno caratteristiche ed implementazioni che non sono stati previsti nelle specifiche del progetto: basti pensare alla possibilità di giocare online offerta dal programma Winboard, cosa che non è prevista nelle specifiche oppure la possibilità di tener conto

delle statistiche personali dell'utente, l'inserimento in una graduatoria online dei propri risultati, documentazioni online ecc.

In conclusione la differenza di risultati è da attribuire principalmente alle differenze tra progetto e programma oltre che, in minor misura, a chi esegue il conteggio: è da tener presente che si sta confrontando un risultato ottenuto da una macchina con un risultato ottenuto da un essere umano.

7. CONSIDERAZIONI PERSONALI

Lo studio del Function Point è stata sicuramente un'esperienza interessante: oltre a farmi capire come ci si deve porre nei confronti del software quando lo si va a misurare, è stato utile per farmi vedere un programma in un'ottica differente. Dovendo valutare dimensionalmente il software sei costretto a porti domande che probabilmente non ti faresti se avessi davanti un programma già finito a cui non devi mettere mano; ti costringe a chiederti come si comporta, quali sono le parti che comunicano tra loro come vengono mostrati i risultati ecc.

Il lavoro di ricerca non è stato semplice: ci sono pochi siti che trattano questo argomento e alcuni di essi sono poco documentati e aggiornati; si sono comunque rivelati utili per conoscere la storia e lo sviluppo dei Function Point e di chi li ha ideati. Le più importanti fonti bibliografiche trattate sono stati i documenti ufficiali, rilasciati attraverso i rispettivi siti web, dall' IFPUG e dall'associazione COSMIC: il primo caso ha richiesto un'iscrizione al sito mentre il secondo concedeva il download del manuale previa la compilazione di un form, il tutto comunque molto semplice da reperire.

I metodi analizzati sono stati interessanti e hanno suscitato in me diverse impressioni che qui di seguito vi spiegherò.

7.1 Sul metodo IFPUG v. 4.2.1

Questo metodo è quello che tra i due mi ha suscitato le migliori impressioni. Ha numerosi punti di forza e, a mio parere, ben pochi difetti ma come ogni cosa creata dall'uomo, è migliorabile.

Punti di Forza:

- Livello di dettaglio elevato
- Manuale spiegato molto bene
- Semplicità nell'applicazione ad un software

Punti di Debolezza:

- Conteggio non immediato

In sostanza, come detto in precedenza, i punti di debolezza son davvero minimi: il conteggio non è immediato ma può essere ovviato dalla realizzazione di una tabella simile a quella da me utilizzata (par 5.3) e che lo stesso manuale consiglia di usare per semplificare il conteggio.

I punti di forza invece si notano molto specialmente se lo si utilizza prima del metodo COSMIC: il livello di dettaglio che fornisce il manuale dando spazio a molteplici opzioni rende l'identificazione degli attributi molto semplice ed immediata. Il manuale ufficiale è davvero ben strutturato e lascia ben pochi spazi alle domande grazie alla completezza delle informazioni in esso contenute; la cosa che più mi ha colpito è stata la specificità delle istruzioni.

In conclusione direi che è risultato semplice lavorare con questo metodo.

7.2 Sul metodo COSMIC v. 2.2

Ben altra storia è risultato il metodo proposto dal gruppo COSMIC.

Punti di Forza:

- Adatto a specifici tipi di software
- Manuale tradotto in diverse lingue
- Conteggio semplice e lineare

Punti di Debolezza:

- Formula di conversione dettata dall'esperienza
- Poco dettagliato
- Spiegazioni dispersive e poco mirate
- Di non semplice applicazione

Il metodo COSMIC non mi ha attratto particolarmente, probabilmente anche perché dovuto al fatto che è stato davvero complicato identificare le diverse parti richieste dal metodo (Enter, eXit, Read e Write su tutto il resto). A differenza del metodo IFPUG, questo non presenta un livello di dettaglio tale da poterlo immaginare

applicato a molteplici software: le spiegazioni ci sono ma non sono focalizzate al raggiungimento di un valore numerico, obiettivo di questi metodi. Il manuale è distribuito in diverse lingue e questo indubbiamente semplifica la vita per chi non è abile con la lingua inglese, ma ha comunque voglia di conoscere questo metodo; un altro punto di forza è sicuramente il conteggio che è molto lineare e semplice da eseguire (non ci sono calcoli da fare se non una semplice somma), una volta identificati i vari aspetti del software.

In conclusione direi che questo metodo a si aspetti positivi, ma ha ancora molto da imparare dal metodo IFPUG.

7.3 Sul metodo SLOC

Punti di Forza:

- Molto pratico
- Conteggio immediato

Punti di Debolezza:

- Richiede l'utilizzo di programmi di terze parti
- Fattori di conversione derivati dall'esperienza

Questo metodo è sicuramente il meno affidabile tra i tre metodi presentati e questo perché non è frutto di ragionamenti e regole ma perché valutato secondo basi empiriche. Il fatto che il fattore di conversione sia determinato dall'esperienza, rende il valore non del tutto certo, opinabile e di conseguenza poco affidabile. Resta comunque un metodo molto utile perché in pochi passi si riesce a fare una stima di quale sia il costo di un software e rende comunque confrontabili due progetti di cui si hanno fonti differenti (codice sorgente, diagrammi UML) seppur con le perplessità espresse in precedenza.

7.4 Considerazioni Finali

In conclusione direi che, come detto in precedenza, il metodo IFPUG è il più completo e facilmente utilizzabile. Un aspetto molto importante sul quale ancora non mi sono espresso è l'esperienza che deve essere maturata di chi si pone l'obiettivo di

contare i Function Point di un software: come detto nel paragrafo 2.2.3, uno dei problemi di applicazione è proprio il fatto che chi non ha esperienza nel conteggio avrà difficoltà ad utilizzare questi metodi.

Per me è stata la prima esperienza e devo concordare con quanto consigliato dal manuale: non è per niente semplice o immediato valutare un software senza aver studiato il funzionamento di questi metodi ma soprattutto senza aver visto come si mettono in pratica i vari passi del manuale. Questa considerazione vale sia per il metodo IFPUG sia per il metodo COSMIC che come già detto non è nemmeno immediato come metodo e quindi richiede un ulteriore sforzo per essere applicato in maniera corretta. Con un buon livello di conoscenza dei metodi studiati e una discreta esperienza applicativa su diversi tipi di software penso che entrambi i metodi possano essere ritenuti efficaci e molto utili per la valutazione di un software, aspetto che ritengo debba essere sempre più ricercato dalle aziende che cercano di emergere in un mercato competitivo come quello dei giorni nostri.

BIBLIOGRAFIA

- [1] R. Traversi, "Analisi e sviluppo di un programma di Kriegspiel", Tesi di laurea, 2006.
- [2] P. Ciancarini, "Project management dei progetti software", Appunti Lezione, 2006.
- [3] R. Meli, "Function Point estimation methods: a comparative overview", Articolo, 1999.
- [4] IFPUG, "IFPUG 4.1 Unadjusted functional size measurement method", Manuale, 2000
- [5] A.J. Albrecht, "Function Point Analysis", Encyclopedia of Software Engineering, 1994
- [6] A.J. Albrecht, J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", 1983
- [7] A.J. Albrecht, J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", Institute of Electrical and Electronics Engineers, 1983
- [8] A.J. Albrecht, J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", McGraw-Hill, 1993
- [9] A.J. Albrecht, J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation," IEEE Transactions on Software Engineering, vol. 9, no. 6, Dicembre 1983
- [10] A.J. Albrecht, J.E. Gaffney, "Software Function, Source Lines of Code, and Development Effort Prediction: A Software Science Validation", IEEE Press, 1983
- [11] MPC COSMIC, Manuale di Misurazione COSMIC-FFP 2.2, Guida implementativa COSMIC a ISO/IEC 19761:2003, gennaio 2003
- [12] A.J. Albrecht, "Function points as a measure of productivity," in Proc. GUIDE 53 Meeting, Dallas, TX, Novembre 1981.
- [13] C.R. Symons, "Function Point Analysis: Difficulties and Improvements," IEEE Transactions on Software Engineering, vol. 14, no. 1, Gennaio 1988

[14] A.J. Albrecht, "AD/M Productivity Measurement and Estimate Validation-Draft," IBM Corp. Information Systems and Administration, AD/M Improvement Program, Purchase, NY, Maggio 1984.

[15] [1] A.J. Albrecht, "Measuring application development productivity." inProc. IBM Applications Development Symp., GUIDE Int. and SHARE Inc., IBM Corp., Monterey, CA, Ottobre 1979.

WEBBOGRAFIA

- [1] D. Natale, Descrizione Metriche,
<http://www.mondomatica.it/function.htm>, gennaio 2008
- [2] D. Natale, 2004,
http://www.cnipa.gov.it/HTML/newsletter/n3_2004/pag.%206.htm, gennaio 2008
- [3] J. Virta, Function Points from UML Specifications,
<http://web.abo.fi/~kaisa/HV.pdf>, novembre 2007
- [4] COSMIC-FFP method approved as an ISO standard,
<http://www.totalmetrics.com/cms/servlet/main2?Subject=Content&ID=583>, ottobre 2007
- [5] C. Symons, "Come Back Function Point Analysis",
<http://www.gifpa.co.uk/library/Papers/Symons/fesmafpa2001/paper.html>, novembre 2007
- [6] COSMIC Full Function Point,
<http://www.dpo.it/topics/methods/cosmic-ffp.htm>, dicembre 2007
- [7] Wikipedia, "C++", 2008
<http://it.wikipedia.org/wiki/C%2B%2B>, febbraio 2008
- [8] F. Vogelezang, "Implementing COSMIC-FFP as a replacement for FPA",
<http://www.gelog.etsmtl.ca/publications/pdf/829.pdf>, febbraio 2008