Università degli Studi dell'Insubria

FACOLTA' DI SCIENZE MM.FF.NN. - VARESE

Corso di Laurea Magistrale in INFORMATICA

Open BQR: una proposta per la valutazione del software Open Source

Tesi di Laurea di:

DAVIDE TAIBI

Matricola:

617337

Relatore:

Prof. LUIGI LAVAZZA

Correlatore:

Prof. SANDRO MORASCA

Anno Accademico 2005-2006

Sommario

Capitolo

1

INTRODUZIONE

Tra i fenomeni significativi legati allo sviluppo delle ICT sta assumendo particolare rilievo quello che va sotto il nome di Software Open Source(OSS).

Per molti anni ha avuto diffusione limitata, soprattutto tra gli sviluppatori, nelle università e negli enti di ricerca. In seguito, con la nascita di numerose aziende distributrici di software a codice sorgente aperto, il modello Open Source si è diffuso in diversi Paesi del mondo.

Il software “libero” negli ultimi anni è diventato oggetto di un duro scontro commerciale e istituzionale, tra Stati Uniti e Unione Europea, fra i produttori di hardware e software proprietario e gli sviluppatori dello stesso software libero. Inoltre la stessa disputa è emersa tra le corporations che detengono i diritti del software commerciale e i governi che invece, vogliono valorizzare e incentivare l’uso del software libero.

La rilevanza internazionale assunta dal fenomeno free software – Open Source, ha indotto il Ministero per l'Innovazione e le Tecnologie a promuovere uno studio sul software a codice sorgente aperto, al fine di consentire una corretta valutazione delle possibilità d'utilizzo nella pubblica amministrazione.

Il motivo di tale importanza sta nel fatto che il software è alla base dell’automazione di ogni produzione tecnologicamente avanzata e di tutta la moderna industria, dal controllo dei flussi produttivi dell’industria manifatturiera alla convergenza multimediale del mondo dei media e della comunicazione, dall’innovazione della Pubblica Amministrazione, all’uso personale di Internet per istruirsi, per lavorare e comunicare a distanza.

Nella maggioranza dei prodotti software vale il principio dei costi della “prima copia”, dovuti agli investimenti spesi per ideare, progettare e realizzare il “numero uno”. Il fattore legato alla distribuzione incide minimamente sul costo totale.

Di conseguenza, la diminuzione dei costi marginali e quindi dei margini di redditività per l’impresa, risiedono nella possibilità di conquistare un vasto mercato, guadagnando così economie di scala dal lato della domanda.

Esistono però alcuni prodotti che non presentano le possibilità sopra descritte: vi sono alcuni programmi per cui il costo di manutenzione è di gran lunga superiore a quello della licenza.

Conseguentemente si possono individuare due principali categorie di utenti:

- Acquirenti *support-oriented*: coloro che percepiscono un’utilità extra da servizi aggiuntivi e supporto forniti con l’acquisto del software, quale manutenzione, supporto tecnico, assistenza…
- Acquirenti *support-indipendent* che non derivano alcuna utilità extra da servizi aggiuntivi e supporto.

Nel primo caso, la creazione del software Open Source porterebbe sicuramente ad una fidelizzazione del cliente, condizionato poi a far fronte ai costi di assistenza.

Il rovescio della medaglia può essere spiacevole: un’azienda che commercia prodotti Open Source dà la possibilità ai propri concorrenti di fornire assistenza all’utente sul loro prodotto senza pagare alcuna royalty, che da un punto di vista prettamente economico, è da considerarsi come un ampliamento del mercato locale.

Un’auspicata evoluzione e la conseguente distribuzione del software OS potrebbe determinare una serie di vantaggi in termini di contenimento dei prezzi, trasparenza e sicurezza, non dipendenza da un unico fornitore, elevata ricusabilità ed accessibilità per le piccole realtà di sviluppo.

Infatti la maggior parte dei prodotti Open Source disponibili, accusa difetti di maturità. L'essenziale per risolvere il problema è l'individuazione dei parametri che definiscono un software maturo.

La vera domanda che ci si pone al momento dell'adozione di un prodotto O.S., è basata su un solo requisito:un sistema in grado di soddisfare al meglio le esigenze dell'utilizzatore. Lontano da essere una vaga domanda esistenziale, la questione è estremamente pragmatica, completamente localizzata e inutilmente senza risposta.

Contesto applicativo

Il presente lavoro di tesi si colloca in un filone di attività che corrispondono ad interessi comuni del progetto QualiPSo (Qualità Platform for Open Source Software), un progetto integrato con l'intenzione di dare un miglior contributo allo stato dell'arte del software Open Source definendo e implementando tecnologie, procedure e policies per influenzare le attuali pratiche di sviluppo Open Source.

Pertanto, i risultati ottenuti da questo lavoro di tesi, saranno propedeutici al lavoro da svolgere nel progetto.

Attualmente, la scelta di un prodotto Open Source è basata solamente sulle esperienze professionali del manager, con i conseguenti possibili rischi di valutazione soggettivi non supportati da alcun parametro oggettivo.

Considerando applicazioni relativamente piccole, l'esperienza può rivelarsi molto efficiente, sia in fattori di costo che di risultati ma, parlando di applicazioni di dimensioni elevate, la questione può diventare molto complessa.

Obiettivi della tesi

Scopo di questa tesi è la formulazione di un modello formale semplice ed intuitivo per la comparazione, la qualificazione e la selezione di prodotti Open Source semplice, rapido ed affidabile.

Si intende creare un modello in grado di assistere i top manager nell'adozione di nuovi prodotti e nella scelta del miglior prodotto disponibile tra un ampio ventaglio di opzioni.

Il percorso di analisi sarà composto dapprima dalla formulazione del modello, quindi dalla verifica della sua validità attraverso due casi pratici: una comparazione tra prodotti CMS (Content Management System) per la realizzazione si applicazioni web, e la valutazione di Compiere, l'ERP Open Source al momento più utilizzato.

Approccio al problema

La natura stessa del progetto comporta la necessità di interagire contemporaneamente con diverse attività più o meno complesse, che possono richiedere un accesso attraverso percorsi del tutto differenti.

Il problema viene affrontato in modo razionale, valutando prima i principali metodi di stima presenti in letteratura, estrapolando quindi i concetti necessari al raggiungimento dello scopo ed aggiungendo infine le parti mancanti.

Struttura della tesi

In questo capitolo è stata chiarita la teoria sulla quale si basa tutto il lavoro di tesi, indicando gli obiettivi principali da raggiungere, le possibili difficoltà incontrabili, nonché lo scenario e le tecnologie di riferimento per lo sviluppo del lavoro.

Nel secondo capitolo vengono analizzate le applicazioni Open Source e, nello specifico, i sistemi principali ERP Open Source.

Nel terzo capitolo vengono introdotte, in modo critico, le metriche del software classiche e ad oggetti analizzandone l'utilizzo da parte della comunità del Software Metrics attraverso un sondaggio distribuito su larga scala.

Nel quarto capitolo vengono illustrate dettagliatamente le metriche del software Open Source esistenti.

Il quinto capitolo contiene la formulazione del modello di valutazione per prodotti software Open Source quindi, nel sesto capitolo vengono introdotti due casi di misurazione reali: il primo volto a comparare tre prodotti CMS, il secondo un ERP Open Source.

Infine vengono tratte le conclusioni del lavoro svolto e i possibili sviluppi futuri.

Si riportano quindi in appendice le tabelle riassuntive per tutti i metodi di stima trattati, la descrizione dettagliata delle metriche di prodotto calcolate su Compiere e i risultati ottenuti da un sondaggio effettuato alla comunità del Software Metric.

Capitolo

2

SISTEMI OPEN SOURCE

American Airlines fu la prima azienda mondiale a vantarsi dell'ausilio di un sistema informativo informatizzato per la gestione della prenotazione dei voli. Il sistema, nel corso degli anni, la cui prima versione è risalente al 1962, è stato più volte modificato ed aggiornato, rimane alla base della gestione delle prenotazioni utilizzato dalla Sabre Holding.

Naturalmente l'avvento delle nuove tecnologie ha permesso di compiere numerosi passi avanti nella gestione e l'automatizzazione dei sistemi informativi.

Sistemi informatici aziendali

Letteralmente un **sistema informatico aziendale** è "quell'insieme di soluzioni informatiche che supporti e risolva molte, se non tutte, le necessità di automatizzazione e gestione delle informazioni aziendali".

Analizzando i diversi sistemi informativi aziendali per la loro applicazione, si trovano numerose classificazioni come gli **ERP**, dall'acronimo dell'espressione Inglese Enterprise Resource Planning, che presenta una versatilità applicativa decisamente molto ampia oltre che ad una rilevanza dominante nel suo settore. Questo sistema se visto dal lato puramente costruttivo è stato implementato per semplificare la gestione delle risorse produttive anche se, nel corso dei tempi, la sua settorialità si è notevolmente ampliata.

Questo tipo di sistema è in grado di gestire le transazioni informative aziendali senza l'ausilio di strumenti informatici e pone maggiore accento all'ottimizzazione dell'uso delle risorse disponibili e di ogni settore legato al ciclo di vita di un azienda produttiva.

Evoluzione

Da una analisi storica della sua evoluzione i precursori diretti del sistema ERP sono le metodologie **MRP** (Material Requirements Planning), **MRP_II** (Material Resource Planning).Attualemente l'ottimizzazione delle procedure utilizzate dalla metodologia MRP II sono al centro dell'interesse delle aziende e delle imprese di differenti settori.

Sistemi informatici aziendali Open Source

Esistono numerosi esempi di sistemi informatici aziendali come SAP, Oracle, IBM, Microsoft, Baan, PeopleSoft, Hyperion. Si tratta però di applicazioni commerciali con l'ausilio di strumenti proprietari. Negli utlimi anni, va precisato, che si è cercato di avvalersi di standard comuni che permettono lo scambio di dati fra le differenti soluzioni.

Il mondo Open Source

Nella ultima decade l'Open Source ha avuto un'enorme espansione, grazie naturalmente alla nascita di prodotti aperti, come ad esempio Linux, anche se il settore aziendale, è stato toccato dalla sua scia in forte ritardo.

Fra i sistemi informatici aziendali Open Source, il cavallo di battaglia è Compiere, il sistema più maturo ed affidabile, ma altri progetti, come SQL-Ledger ed ERP5, si trovano sulla buona strada anche se necessitano di ulteriori miglioramenti.

Perché

Uno dei cardini importanti per dar vita ad un progetto è la fiducia in quello che si sta per fare; il mondo open source inizialmente non dava molte garanzie e conseguentemente

ha ritardato la nascita di nuove valide alternative. La situazione è stata aggravata ulteriormente dalla mancanza di un corretto modello imprenditoriale capace di dare soluzioni efficienti ed affidabili. Nell'ultimo periodo sono stati introdotti nuovi modelli di business capaci di attirare l'attenzione delle PMI ma anche delle grandi aziende, attraverso interessanti possibilità economiche.

Il modello imprenditoriale

Il modello imprenditoriale per il momento più all'avanguardia è senza dubbio Jboss, rilevatosi un modello molto semplice implementato dalla JBoss Inc, che ha sviluppato un server applicativo J2EE, il JBoss Application Server.

Compiere

Si tratta di un server applicativo 100% Pure Java che si avvale dell'ausilio di Oracle 9i come DBMS, sviluppato dalla ComPiere Inc. Compiere utilizza anche l'infrastruttura concessa da server applicativi ben più complessi ,come JBoss e Tomcat.

ERP5

Purtroppo anche se si tratta di un progetto interessante ed innovativo, simile a Compiere, non è ancora giunto alla piena maturità, forse anche per il fatto di essere molto recente.

Il motivo della scelta di sviluppare un ulteriore server è legata al fatto che all'epoca della nascita di ERP5, Compiere non era ancora completo, sostanzialmente era sprovvisto di documentazione.

ERP5 è comunque un server destinato a crescere con un futuro applicativo molto ampio in quanto sistema informatico molto promettente, data la sua infrastruttura semplice e leggera di classi (solo cinque).

SQL-Ledger

Si tratta di un sistema informatico per la gestione della contabilità a partita doppia., multipiattaforma, viene comodamente istallato ed eseguito in una miriade di differenti architetture, buona parte delle distribuzioni Unix, Linux e delle famiglie di sistemi operativi Microsoft Windows e Mac OS X incluse. Il linguaggio di programmazione è il Perl e non è necessario utilizzare il codice sorgente, ciò garantisce un livello elevato di prestazioni e di affidabilità nella gestioni dei dati relazionali.

L'interfaccia utente è implementata in html ed è possibile utilizzare diversi DBMS, come Oracle, PostgreSQL o DB2.

Vantaggi e svantaggi

Un vantaggio che sicuramente contraddistingue ERP5 è la natura costruttiva del sistema, ERP5 è nato come sistema Open Source, mentre l'applicazione originale di Compiere era quella di software proprietario.

Il linguaggio di programmazione utilizzato nello sviluppo di ERP5 è Python, la sua infrastruttura è quella del server applicativo Zope. Python, rispetto a Java, è in grado di utilizzare lo stesso linguaggio di programmazione nello sviluppo, nell'infrastruttura applicativa, nel linguaggio di scripting per la personalizzazione delle procedure.

Il primo punto di forza, ma anche di debolezza, di SQL-Ledger è che rispetto ad altri sistemi informatici aziendali Open Source, si propone di risolvere un sottoinsieme relativamente ridotto delle problematiche aziendali in modo rapido ed efficace.

SQL-Ledger ha il grande vantaggio di essere localizzato nativamente in un gran numero di lingue (più di venti), e non solo, offre pure un'ampia scelta di piani contabili già pronti all'uso, senza necessitare di alcun intervento preventivo, se non appunto un'azione di scelta, da parte dell'utente finale.

Dopo averlo testato, oltre ad apparire evidente che in realtà SQL-Ledger è adatto soltanto a realtà aziendali abbastanza piccole, si può anche affermare che un altro dei

punti deboli di SQL-Ledger è il fatto che non è molto ben documentato, sia per quanto riguarda l'utente finale sia che per quanto riguarda lo sviluppatore. Se questo è vero fino ad un certo punto anche per Compiere, bisogna però aggiungere che quest'ultimo, data una maggiore somiglianza con alcuni sistemi informatici già ampiamente utilizzati, consente un approccio più intuitivo all'utente finale, e d'altra parte, per la maggior parte degli sviluppatori, il codice sorgente scritto in Java è molto più autoesplicativo di quello scritto in Perl.

Dal punto di vista di uno sviluppatore, uno dei vantaggi di SQL-Ledger è invece sicuramente il fatto che è un gran bello strumento per imparare che cos'è la contabilità e per studiare le varie implicazioni correlate alle diverse procedure contabili; d'altra parte, un Ragioniere o un Commercialista, che probabilmente al giorno d'oggi utilizzano già un qualche programma per la gestione della contabilità e che si suppone conoscano abbastanza approfonditamente cosa sia la contabilità stessa, potrebbero non vedere alcun vantaggio nell'utilizzo di un tale sistema informatico.

Altri sistemi informatici aziendali Open Source

Naturalmente Compiere, ERP5 e SQL-Ledger non sono che alcuni dei più importanti prodotti ERP, gli strumenti Open Source che promettono di fornire le funzionalità tipiche dei sistemi informatici aziendali sono centinaia, se non migliaia. Tra i vari progetti, un cenno di riguardo merita sicuramente il progetto GNUe, che, come lascia anche supporre il nome, è un ambizioso progetto portato avanti dalla Free Software Foundation. Sul fronte Java, un più diretto concorrente per Compiere, è Value, sviluppato e rilasciato dalla Emryn International LLC. Come SQL-Ledger, il suo ambito operativo è però molto limitato rispetto a quello di Compiere, esso è infatti confinato alla contabilità ed alla gestione delle vendite, SellWin (http://sellwincrm.sourceforge.net/) è un altro progetto basato sulla possibilità di utilizzo "multidevice", sviluppato in Pure Java – XML su architettura multi livello, viene presentato con un interfaccia web standard (HTML) ed un'altra per dispositivi mobili

(J2ME). Ohioedge CRM, JERPA, OPENCRX e molti altri prodotti ERP Open Source come loro, stanno cercando di guadagnarsi una nicchia di mercato specializzandosi in un ambito diverso (contabilità, gestione vendite, gestione clienti…), infine Oratio, nato da una biforcazione del programma di contabilità SQL-Ledger di Dieter Simader, della società italiana Proxima-Centauri. Oratio® è un software che, attraverso la gestione integrata, è in grado di supportare tutti i processi lavorativi aziendali, dalle offerte, all'ordine di vendita o di acquisto, dai movimenti di magazzino al processo di produzione fino alla gestione della contabilità e all'elaborazione del bilancio aziendale in tutti i suoi aspetti. Può essere usato dalle piccole, medie e grandi imprese in quanto è uno strumento potente perchè nato dalla collaborazione di un gran numero di esperti in diversi settori. Il software è semplice da utilizzare, e sono presenti guide di installazione e un breve manuale gratuito.

Capitolo

3

SOFTWARE METRICS

Nel panorama ingegneristico l'industria del software si colloca in una posizione di considerevole importanza.

La complessità del software cresce esponenzialmente grazie a nuovi tipi di input utente, database interattivi, grafica 3d, networks ma, come per ogni cosa che abbia un riscontro pratico, ogni miglioramento deve essere rilevato e misurato. Generalmente la comparazione degli hardware viene sostanzialmente basata su test di benchmark anche se, per il software non sono disponibili test di produttività accettabili. Le comparazioni sono generalmente basate su molteplici metodi di conteggio matematici.

È certamente chiaro che l'aumento della complessità di un software influisce negativamente sulla sua scalabilità, lo sviluppo di sistemi complessi non sempre si traduce in sistemi ben scalabili. L'introduzione della programmazione Object Oriented ha fornito un consistente miglioramento nella programmazione semplificando le fasi di progetto e di implementazione sfruttando la semplice architettura ed una notevole riusabilità di ogni progetto.

La stima delle funzionalità di un prodotto software, il computo dei costi di progettazione, implementazione e manutenzione è ancora nelle fasi primordiali, anche se di contro nessun utente si meraviglia davanti a concetti come tassi di mortalità medi dei progetti software e ed il rincaro sproporzionato dei costi finali rispetto a quelli preventivati.

Misurare per conoscere

I metodi di misura di funzionalità di un software per rappresentare un indice di efficienza e qualità, devono essere indipendenti dal livello tecnologico adottato per la loro realizzazione. Il concetto di “misura del software” è entrato ormai nella consuetudine comune per chi si occupa di progettazione al punto tale da ritenere inefficiente ed incompleto un prodotto che non la riporti. Il processo di misurazione del software, presenta numerose sfaccettature e dipende da svariati fattori. Dalla misura si evidenziano chiaramente gli obbiettivi da migliorare. Il problema risiede nella scelta degli obbiettivi, in base alla loro tipologia o applicazione nel mercato.

Il panorama offre diversi approcci legati alla stima della dimensione funzionale, al miglioramento di alcuni obbiettivi e alle funzionalità rivolte all’utente.

Analisi storica della misurazione del software.

La prima occasione ufficiale di discussione sulla misurazione del software, fu la prima conferenza NATO sul Software Engineering nel 1968.

La prima, misura del software è storicamente rappresentata dal numero di linee di codice (LOC, Lines Of Code); il primo tentativo di calcolo della produttività dei programmatori in termini di numero di LOC per mese-persona risale al 1974.

La misura delle LOC è tuttora utilizzata in contesti industriali significativi, alcuni problemi associati sono stati individuati già dai primi anni ’80 e confermati, motivando la ricerca di metriche alternative.

Dagli anni ’70 sono nate proposte specifiche per la misurazione della dimensione tecnica e della complessità del software quali il coefficiente ciclomatico di McCabe (1976) e la software science di Halstead (1977).

Tra la fine degli anni ‘70 e la prima metà degli anni ’80 sono dunque sorte varie proposte di misura funzionale, intesa come misura indipendente dal linguaggio di

programmazione e da altri aspetti tecnico-implementativi come la misura FFP (Files, Flows, Processes) nel 1983 e la prima versione dei Function Point nel 1984.

Rispetto alle LOC, le metriche funzionali mostrarono una migliore correlazione con i costi di realizzazione, oltre a una maggiore applicabilità in fasi alte del progetto software, decretando quindi un fondamentale passaggio di paradigma da misure "fisiche" o "tecniche" a misure "logiche" o "funzionali" per la stima anticipata del software e la negoziazione contrattuale.

Nel corso di un ventennio la Function Point Analysis, standardizzata dall'IFPUG a partire dal 1986, è evoluta generando molte varianti e proposte di miglioramento, più o meno diffuse quali: Bang Metric, Feature Point, Function Point Mark II (poi confluito nel COSMIC), Function Point NESMA, Function Point 3D, Micro Function Point, Full Function Point.

Nel 1998 è nato il Common Software Measurement Consortium Group, da cui viene pubblicato il metodo dei COSMIC Full Function Point, approvato come metodo standard dall'ISO/IEC nel 2003, insieme ai precedenti metodi proposti da NESMA, Mark II e IFPUG (unadjusted).

In Italia, la misurazione del software è oggetto di studio da tempo. Dai primi anni '90, essa è parte integrante degli impianti di sviluppo e dei contratti di fornitura di grande rilievo, come per la pubblica amministrazione: la prima indicazione dell'Autorità per l'Informatica nella Pubblica Amministrazione (AIPA, ora CNIPA) di adottare la metrica dei Function Point è contenuta in un parere di congruità del 1993 e ha avviato un processo di adozione e diffusione senza soluzione di continuità.

Metriche Di Complessità

Complessità Ciclomatica

La Complessità Ciclomatica (McCabe's Cyclomatic Complexity[8]) è usata per valutare la complessità di un algoritmo in un metodo ed è interamente basata sulla struttura del grafo che rappresenta l'algoritmo.

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

Riferendosi ad un grafo che rappresenta l'algoritmo e posti:

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

si ha:

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

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

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

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

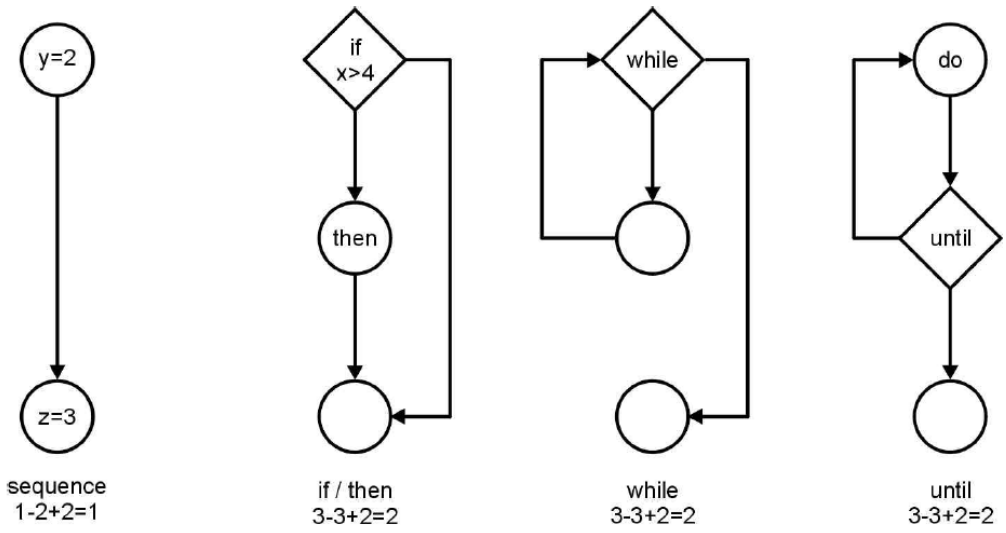


Figura 1: esempio di calcolo della CC per quattro strutture di programmi

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

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

Così come proposto da McCabe[8], la CC di un metodo dovrebbe essere inferiore a 10, in quanto per CC maggiori di 10 si ha un aumento discontinuo degli errori per linea. Nella pratica sono stati utilizzati con successo CC fino a 15 anche se la gestione di questo tipo di progetti richiede una notevole preparazione.

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

La CC ha diverse proprietà:

1. v(G) >= 1
2. v(G) = numero massimo di percorsi linearmente indipendenti in G.

3. L’inserimento o l’eliminazione di espressioni in G non influenza v(G).
4. G ha un solo percorso solo se v(G) = 1.
5. Inserendo un nuovo arco in G, v(G) aumenta di 1.
6. v(G) dipende solo dalla struttura decisionale di G.

Si può dimostrare che la CC di un programma è uguale al numero di nodi decisionali più 1: v(G) = n° decisioni + 1

La complessità ciclomatica può essere utilizzata in svariate aree quali:

- *Code development risk analysis*: durante lo sviluppo software, si può misurare la complessità per valutare i rischi inerenti
- *Change risk analysis in maintenance*: la complessità del codice tende ad aumentare se la manutenzione diventa troppo invasiva. Misurando la complessità prima e dopo una proposta di cambiamento, è possibile monitorarne i rischi relativi.
- *Test Planning*: analisi matematiche, hanno mostrato che la complessità ciclomatica da l’esatto numero test necessari per valutare ogni punto decisionale in un programma per ogni risultato. Conseguentemente, un modulo molto complesso potrebbe richiedere una quantità proibitiva di tests, da tale indice sarà quindi possibile comprendere la necessità di spezzare il modulo in più parti.
- *Reengineering*: L’analisi della complessità ciclomatica, fornisce la conoscenza della struttura del codice di un’applicazione. I rischi correlati alla reingegnerizzazione del codice sono fortemente correlate alla sua complessità.

Punto forza della Complessità Ciclomatica, è la possibilità di essere calcolata tramite strumenti automatici, in grado di ottenere risultati grafici e numerici rapidamente.

La complessità ciclomatica, risulta moderatamente sensibile ai cambiamenti di programma conseguentemente, si ritiene pertanto necessario l'utilizzo contemporaneo di altre metriche.

Metodi di misurazione funzionale

Il metodo di misurazione della dimensione funzionale (FSM) misura la visione logica esterna del software dal punto di vista degli utenti, valutando la quantità di funzionalità da consegnare. La capacità di quantificare in maniera precisa le dimensioni del software, sia nella fase iniziale che in quella finale del ciclo di vita evolutivo del software, è determinante per chi gestisce i progetti di software e deve assicurare una valutazione dei rischi, lo sviluppo delle stime (per esempio, impegno lavorativo, costo del progetto) e la possibilità di disporre anticipatamente degli indicatori del progetto (es. produttività).
La dimensione funzionale è inoltre indipendente dall'effort necessario allo sviluppo o alla manutenzione, dalle metodologie impiegate, dai supporti fisici utilizzati e dalle componenti tecnologiche. Un metodo FSM dovrebbe indicare il grado di convertibilità con altri metodi dimensionali. Altre parti dello standard sono in produzione relativamente a: conformità, verifiche, modello di riferimento, domini software.

La definizione dei concetti relativi alla misurazione della dimensione funzionale del software (FSM) e la descrizione dei principi per applicare un metodo FSM sono contenuti nello standard ISO/IEC 14143-1 del 1997 "Information Technology - Software measurement - Functional size measurement - Definition of concepts".

Un metodo FSM dovrà avere le seguenti caratteristiche:

- si basa su una rappresentazione dei requisiti utente vista da una prospettiva degli utenti;
- può essere applicato tempestivamente appena i requisiti funzionali utente sono stati definiti e sono disponibili;

- deriva la dimensione funzionale a partire da requisiti funzionali, indipendentemente dai requisiti tecnici e dalla qualità.

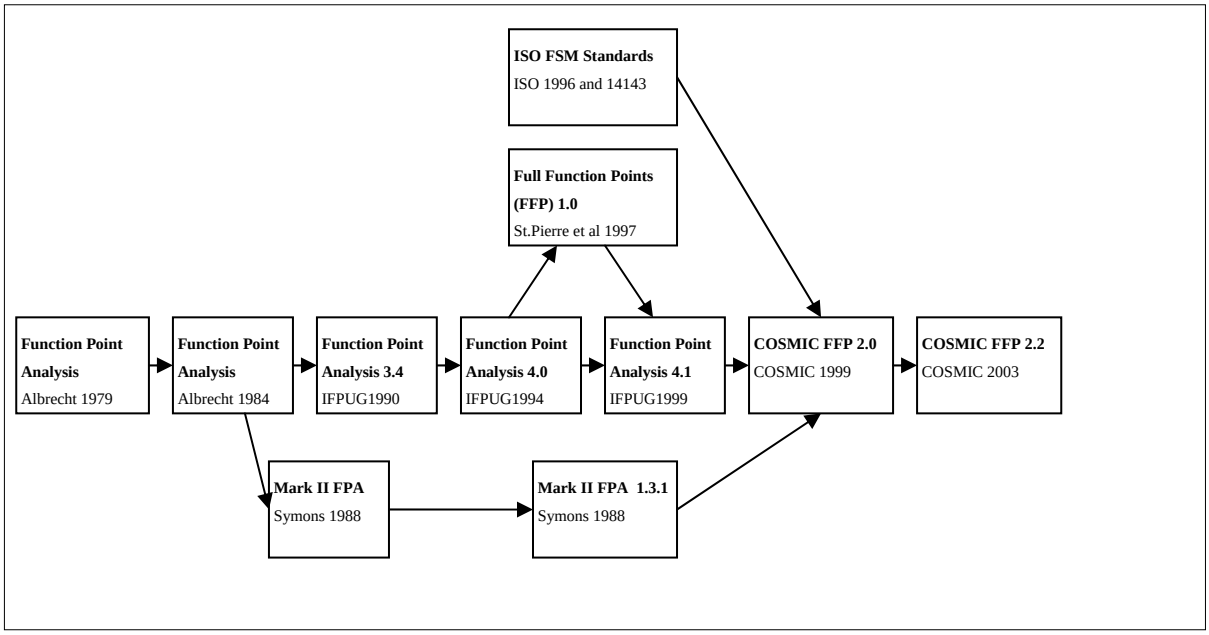


Fig x.x: *Sviluppo dei metodi si stima funzionale*

Il metodo di misurazione della dimensione funzionale più usato è quello noto come IFPUG Analisi dei Function Point (IFPUG FPA) che si basa su un metodo proposto da Alan Albrecht. Questo metodo può essere considerato il precursore dei metodi FSM. Questa tecnica venne messa a punto per misurare la quantità di dati a cui accede una singola funzione come indicatore della dimensione funzionale. Tuttavia questo metodo venne sviluppato per misurare i Sistemi Informativi di Gestione (MIS) e presuppone l'uso di metodologie di sviluppo di software tradizionali come l'analisi e la progettazione strutturata. Sebbene l'IFPUG, Analisi dei Function Point, abbia ottenuto una fama crescente nell'industria, la mancata applicabilità a tutti i tipi di software e la rapida evoluzione dei paradigmi di sviluppo hanno portato alla nascita di numerose variazioni in questo metodo. Per superare queste diversità il COSMIC-FFP ha introdotto la seconda generazione di metodi FSM, applicabile a diversi domini di software, ai moderni concetti di ingegneria di software.

I Function Point

L'uso dei function point come misura della grandezza funzionale del software è cresciuto nel decennio trascorso, passando da poche organizzazioni interessate ad una consistente lista di organizzazioni nel mondo.

L'evoluzione

Verso la fine degli anni '70, Allan Albrecht dell'IBM definì i concetti che permettono di misurare l'entità dei progetti di sviluppo software. Queste definizioni sono state esposte in *IBM CIS & A Guideline 313, AD/M Productivity Measurement and Estimate Validation*, pubblicato il 1 novembre 1984.

Col diffondersi dell'uso dei function point si è avuta un'applicazione sempre più ampia di tale metrica. Ciò ha permesso di effettuare dei test sulla descrizione originale della metrica e ha reso necessaria la creazione delle linee guida per interpretare le regole originali in nuovi ambienti. Come conseguenza è stata realizzata la Versione 2.0 (Aprile 1988) dell'*International Function Point Users Group (IFPUG) Function Point Counting Practices Manual*. Poco dopo (Aprile 1990) è stata pubblicata la versione 3, una delle pietre miliari nell'evoluzione della misurazione della dimensione funzionale. Per la prima volta il Comitato dell'IFPUG per le Regole del Conteggio si è impegnato nel passaggio da una serie di varie interpretazioni delle regole a un documento coerente che rappresentasse un punto di vista comune sulle regole di conteggio dei function point. In tal senso è stato fatto il primo passo per stabilire degli standard sulle misure dei function point che ne permettessero l'applicazione in organizzazioni diverse. La versione 4.0 (Function Point: Manuale sulle Regole di Conteggio, IFPUG, Gennaio 1994) ha costituito un ulteriore pietra miliare nell'evoluzione della misurazione della dimensione funzionale. Questa versione rispecchiava l'utilizzo dei function point come stima della grandezza di un progetto, sin dalle fasi preliminari dello sviluppo, attraverso l'uso delle tecniche di ingegneria del software. La crescita rapida del numero di applicazioni con finestre ad interfaccia grafica è stata una spinta ad includere in tale

versione il conteggio delle GUI (Graphical User Interface). Poiché le attività di conteggio incontravano una maggiore varietà di situazioni, questa versione ha posto enfasi sull'interpretazione e sull'applicazione pratica delle regole di conteggio. Alla documentazione sono stati aggiunti vari esempi e casi di studio supplementari. Successivamente, la versione 4.0 proseguiva nel chiarire il conteggio dei function point e nel migliorarne la coerenza.

- l'identificazione di un utente, di un processo elementare, e delle informazioni di controllo;
- la differenziazione tra Output Esterni (EO) e Interrogazioni Esterne (EQ);
- l'identificazione dei Tipi di Elementi Dati (DET) e dei Tipi di Elementi Record (RET) per le funzioni di tipo dati;
- l'identificazione dei Tipi di Elementi Dati (DET) per le funzioni di tipo transazionale.

Infine, la versione 4.1 continuava il processo di chiarimento e di miglioramento della coerenza del conteggio dei function point e, ad eccezione delle 14 Caratteristiche Generali del Sistema, la versione è stata progettata per essere conforme agli standard ISO. La versione 4.2 non modifica alcuna regola precedentemente emessa, ma fornisce, per le regole esistenti, chiarimenti e interpretazioni evolute, inserendo nei manuali di conteggio esempi e linee guida più chiare.

Caratteristiche

Le caratteristiche principali dei FP sono:

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

Il metodo di calcolo IFPUG

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

Il calcolo dei FP è diviso in più passi:

- Determinazione del tipo di conteggio
- Determinazione dell'ambito e del confine dell'applicazione
- Determinare il numero di Function Point non pesati
    - Conteggio delle funzioni di Tipo Dati (ILF, EIF)
    - Conteggio delle funzioni di tipo Transazionale
- Determinazione del Fattore di aggiustamento del valore
- Calcolo del numero di Function Point pesati

Determinazione del tipo di Conteggio:

Il calcolo dei Function Point può essere eseguito in tre fasi diverse dello sviluppo software:

- conteggio per un progetto di sviluppo,
- conteggio per un progetto di manutenzione
- conteggio per un'applicazione.

Identificazione dell'ambito e del confine dell'applicazione

Prima di calcolare i Punti Funzione occorre Identificare l'Ambito del Conteggio e il confine dell'applicazione, definendo le funzionalità che saranno incluse in un particolare conteggio di Function Point e sottolineando la linea di separazione tra il software oggetto di misurazione e l'utente.

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 funzionalità possono essere incluse in più di una applicazione.

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

Determinazione dei function point non pesati

Il numero di function point non pesati (UFPC-Unadjusted Function Points) riflette il numero di funzionalità specifiche fornite dall’utente dal progetto o applicazione.

Le specifiche funzionalità utente di un’applicazione sono valutate in termini di che cosa viene consegnato con l’applicazione. Solo le componenti richieste e definite dall’utente sono contate.

Il numero di FP non pesati viene definito da due diverse funzioni: dati e transazioni.

Funzioni di tipo Dati

Le funzioni di tipo dati rappresentano le funzionalità fornite dall’utente per soddisfare i requisiti relativi ai dati interni ed esterni. Le funzioni di tipo dati sono costituite sia da file logici interni sia da file di interfaccia esterni.

- Un **File Logico Interno** (ILF) è un gruppo di dati o informazioni di controllo, collegati logicamente e riconosciuti dall’utente, all’interno del confine 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 **File di Interfaccia Esterno** (EIF) è un gruppo di dati o di informazioni di controllo, referenziato dall'applicazione ma mantenuto all'interno del confine di un'altra applicazione oggetto di conteggio. Compito primario di un EIF è di contenere dati referenziati da uno o più processi elementari dell'applicazione che si sta contando. Un EIF contato per un'applicazione deve essere un ILF in un'altra applicazione.

Ad ogni ILF e EIF viene assegnata una complessità funzionale basata sul numero di elementi di tipo dati (Data Element Type, DET) ed elementi di tipo record (Record Element Type, RET).

Regole di conteggio dei DET:

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

- 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
- Conta un DET per ogni dato richiesto dall'utente per stabilire una relazione con un altro ILF o EIF.

Definizione: Un elemento di tipo record o RET è un sottogruppo di elementi di tipo dati riconoscibile dall'utente in un ILF o EIF.

Possono essere opzionali od obbligatori.

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.

Conta un RET per ciascun sottogruppo opzionale o obbligatorio di un ILF o EIF, oppure

• Se non ci sono sottogruppi, conta il ILF o il EIF come un RFT.

In pratica, conta un RET per ogni entità nell'ILF e per ogni relazione con attributi nell'ILF.

Funzioni di tipo Transazionale

Le funzioni di tipo transazionale sono tutte le funzionalità per l'elaborazione dei dati da parte dell'utente.

Vengono definite come input esterni, output esterni o interrogazioni esterne.

- **Input Esterni** (EI – External Inputs): processo elementare in cui i dati attraversano il limite dall' esterno all' interno. Questi dati possono arrivare da una schermata di input o da un ' altra applicazione. I dati sono utilizzati per mantenere uno o più file logici. Una buona fonte di informazioni per determinare gli EI è costituita dagli schemi di schermo, formati di schermo e dialogo, schemi delle maschere di input. Gli input aggiuntivi da altre applicazioni devono essere considerati a questo punto ed aggiorneranno gli ILF delle applicazioni calcolate.
- **Output Esterni** (EO - External Outputs): processo elementare in cui i dati derivati attraversano il limite dall' interno all' esterno. I dati creano dei rapporti o dei file di output inviati ad altre applicazioni. Questi rapporti o file sono creati da uno o più File Logici Interni (ILF). Una buona fonte di informazioni per determinare gli EO sono gli schemi di rapporto e i formati di file elettronici che sono inviati all'esterno dei limiti dell'applicazione.
- **Interrogazione Esterna** (External Inquiry **EQ):** 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 raccolta completa di informazioni sui componenti dei FP andrebbe distribuita tra tutti i componenti del gruppo di lavoro in maniera da assicurarsi che sia stato incluso tutto. Quando si è avuta la conferma che tutte le transazioni ed i file sono stati inclusi si può procedere alla classificazione individuale dei singoli componenti.

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

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

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

Determinare il Fattore di Aggiustamento del Valore

Il fattore di aggiustamento del valore (VAF – Value Adjustment Factor) rappresenta le funzionalità generali fornite all'utente dell'applicazione.

Il VAF è formato da 14 caratteristiche generali del sistema (GSC – General System Characteristics) che valutano le funzionalità generali dell'applicazione. Ogni caratteristica ha una descrizione associata che aiuta a determinare il corrispondente grado di influenza. I gradi di influenza variano in una scala da zero a cinque, corrispondenti a nessuna influenza e a forte influenza.

Le 14 caratteristiche generali del sistema sono:

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

12	Facilità operativa	Quanto efficaci e/o automatici sono le procedure di avvio, back-up e recovery?
13	Installazioni multiple	L'applicazione è stata specificamente progettata, sviluppata e supportata per essere installata su più postazioni e per più organizzazioni?
14	Modificabilità	L'applicazione è stata specificamente progettata, sviluppata e supportata per essere facilmente modificata?

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

Calcolo del numero finale di funcion point pesati

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

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

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

Convalida dei risultati

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

Gli errori più frequenti nel calcolo dei FP sono gli errori di omissione, altri errori sorgono quando costruzioni fisiche sono sostituite da costruzioni logiche e contate come

componenti. Pertanto il gruppo di lavoro dovrà controllare l' analisi dei FP per completezza e verificare che tutte le componenti (EI, EO, EQ, ILF e EIF) siano state incluse.

Il coordinatore delle metriche deve lavorare a stretto contatto con chi esegue materialmente i calcoli per assicurarsi che il processo e la validazione siano stati eseguiti correttamente.

Chi esegue il calcolo dei FP dovrà porsi le seguenti domande:

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

12. Il calcolo è stato registrato in un archivio per i FP?
13. Le premesse sono coerenti con tutti gli altri progetti?
14. Le premesse sono state accuratamente documentate?
15. Il calcolo è stato controllato da un esperto in calcoli di FP (possibilmente certificato)?

Pro Contro

La valutazione dei pesi, avviene effettuata durante la fase iniziale del progetto, sulla base di documenti di specifica o di progetto, spesso del tutto informali e scritti in linguaggio naturale.

Conseguentemente, il conteggio avviene su basi piuttosto fragili, con ampio margine di interpretazione delle specifiche da parte dell'esperto del conteggio.

Inoltre, è possibile verificare che i Function Point, tengono conto delle funzionalità, senza considerare la complessità algoritmica oltre ad effettuare un'eccessiva semplificazione delle tipologie dei componenti: a un sistema con 100DET viene assegnato solo il doppio di un sistema con un solo DET.

La somma dei FP di piccoli sottoprogetti non corrisponde al calcolo dei F.P. dell'intero progetto.

Lo stesso Albrecht afferma che l'utilizzo dei F.P. deve essere usato come ulteriore misura a supporto del pensiero dei manager.

COCOMO

Il COCOMO è un modello creato da Barry Boehm (1981) per dare stime del numero di programmatori-mese necessari per la produzione di un prodotto software.

Il **COnstructive COst Model** si basa sullo studio di sessanta progetti al TRW, una compagnia Californiana di automazione e sviluppo software, acquistata da Northrop Grumman nel tardo 2002. I programmatori hanno esaminato progetti con dimensioni

che vanno dalle 2000 alle 100.000 linee di codice, per linguaggi di programmazione che spaziano dall'assembly al PL/I.

Il calcolo COCOMO è basato sulla stima della dimensione di un progetto in Linee di Codice Sorgente (SLOC) definite come:

- Una SLOC è una linea di codice
- Le dichiarazioni sono considerate SLOC
- Il codice scritto dai software generatori di codice è escluso
- Le linee di commento sono escluse.

Il COCOMO 81, fu definito in termini di Delivered Source Instructions, molto simili alle SLOC. La differenza principale tra le DSI e le SLOC è che una singola riga di codice potrebbe essere composta da più linee. (es. un if – then – else, può essere considerato una SLOC se scritto in una linea sola ma viene considerato più di una linea utilizzando le DSI).

Cocomo consiste in una gerarchia ad albero di dettaglio crescente.

- *Basic COCOMO*: è un modello statico a singolo valore che calcola lo sforzo di sviluppo del software (e il costo) come funzione della grandezza del programma espressa in linee di codice ipotizzate, fornisce una stima veloce ma approssimata e prematura.
- *Intermediate COCOMO*: calcola lo sforzo di sviluppo del software come funzione della grandezza del programma e un insieme di "indici di costi" che includono l'assegnazione soggettiva di valutazioni di prodotti, hardware, attributi di progetto e personali. Fornisce una stima più accurata.
- *Detailed COCOMO*: incorpora tutte le caratteristiche del COCOMO intermedio con una valutazione dell'impatto dei vari costi per ogni passo (analisi, progettazione, ecc.) del processo di ingegneria del software. Fornisce la miglior

stima in quanto tiene conto dei fattori dell'ambiente del progetto software in ogni fase del progetto.

Una delle osservazioni importanti nel modello è che motivazioni personali affiancano tutti gli altri parametri. Ciò significa che le abilità del team e dell'individuo incaricato di tale valutazione sono le più importanti di tutte.

Il concetto base del modello COCOMO è l'uso delle equazioni di effort per stimare il numero di persone-mese richieste per sviluppare un progetto.

COCOMO Basic

Fornisce una precisione di 20% nel 60% dei casi. Lo sforzo E richiesto per lo sviluppo di un sistema software è in funzione della dimensione del codice S espressa in KLOC (migliaia di linee di codice):

$E=a\ S^b$

La durata T del progetto è funzione dello sforzo E : $T=cE^d$.

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

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

Tabella x.x: *Valori delle costanti per il modello Basic*

COCOMO Intermediate

Fornisce una precisione di 20% nel 68% dei casi. Questo modello è basato sul modello Basic al quale introduce 15 fattori correttivi detti Fattori di Complessità (Cost Drivers) o

Moltiplicatori di Sforzo (EM – Effort Multipliers). Ad ogni fattore di costo occorre assegnare un valore come dalla seguente tabella:

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

Tabella x.x: *Fattori di costo*

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

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

dove $EM_{1,\ldots,15}$ sono i valori assegnati ai fattori di complessità (moltiplicatori di sforzo) secondo la tabella. Lo sforzo E richiesto per lo sviluppo di un sistema software è funzione dello sforzo nominale E_{nom}:

E= E nom EAF.

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

La durata T del progetto è funzione dello sforzo E :

$T=cE^d$

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

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

Tabella x.x: Valori delle costanti per il metodo Intermediate

COCOMO Detailed:

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

COCOMO Detailed definisce sei fasi del ciclo di vita:

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

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

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

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

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

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

Pro e Contro

Il modello COCOMO risulta essere molto legato alla conoscenza del dominio e all'abilità di analisi personale.

Al momento della creazione del modello (1981) il campo di utilizzo era completamente diverso, la programmazione web, i linguaggi Object Oriented e ambienti di sviluppo integrati non erano ancora esistenti. Con l'introduzione delle nuove tecnologie, risulta difficile applicare il modello.

COMOMO Basic inoltre, indica dei coefficienti, basati sull'analisi di un largo numero di progetti, ritenuti universalmente applicabili a qualsiasi applicazione esistente. Una

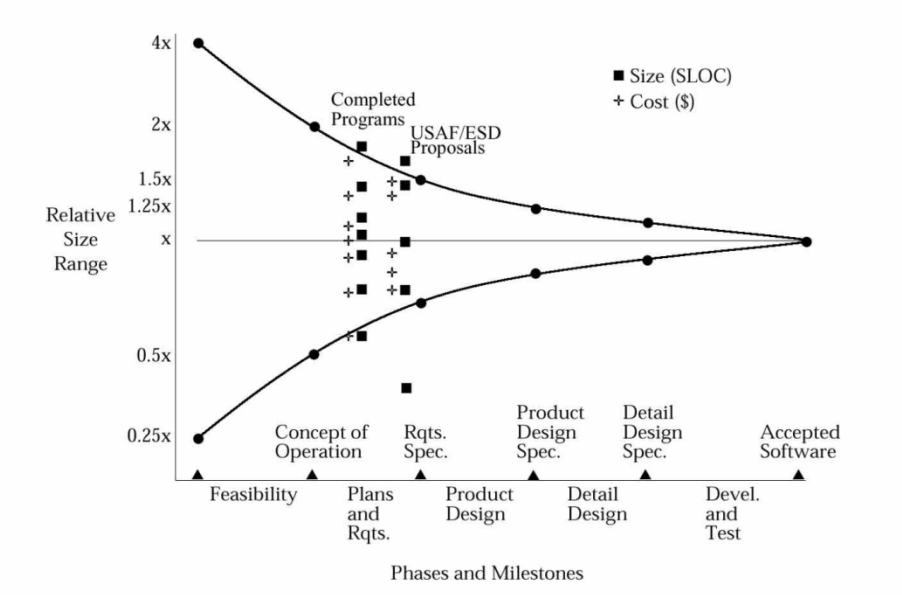
così spinta generalizzazione comporta però la possibilità di situazioni di progetti diversi da quelli presi in considerazione, in cui i coefficienti potrebbero risultare inadatti.

Il passaggio tra il modello Intermediate e Detailed, fornisce un miglioramento del 2% a fronte di un aumento dei tempi di stima notevole.

COCOMO II

Verso la metà degli anni novanta le tecniche di sviluppo del software cambiarono drasticamente. Questi cambiamenti includevano modelli di sviluppo rapidi e non sequenziali, l' enfatizzazione del riuso del software esistente, l' utilizzo di componenti software standardizzate e pronte all' uso, reingegnerizzazione, l' applications composition, l' approccio object-oriented. Questi cambiamenti resero problematica l'applicazione di COCOMO 81, così la soluzione più logica fu quella di reinventare il modello per gli anni novanta. Dopo diversi anni di sforzi combinati tra USC (University of Southern California) – CSE (Center for Software Engineering), IRUS presso UC Irvine ed il COCOMO II Project Affiliate Organizations, il risultato fu COCOMO II, un modello rivisitato per la stima dei costi del software che rifletteva i cambiamenti avvenuti nello sviluppo professionale di software fin dagli anni settanta. COCOMO II, diversamente da COCOMO 81 che usa solo il ciclo di vita a cascata (waterfall), deve adattarsi ai diversi cicli di vita del software ed alle particolarità dei loro processi (disponibilità di software riusabile, grado di comprensione dell' architettura e dei requisiti, vincoli di tempo o di dimensione, affidabilità richiesta,…).

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



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

COCOMO II divide il mercato software in 5 segmenti principali:

- **End-User Programming** (95.24%): programmi piccoli e flessibili, sviluppati dagli stessi utenti attraverso degli Application Generators.
- **Infrastructure** (1.36%): prodotti nell' area dei sistemi operativi, sistemi di gestione dei database, sistemi di networking. I produttori di questo tipo di software, contrariamente ai programmatori end-user, hanno un' ottima conoscenza della computer science ed una conoscenza scarsa delle applicazioni.
- **Application Generators and Composition Aids** (1.09%): prodotti prepackaged per user programming. Questi prodotti hanno molti componenti riusabili ma richiedono comunque ottime capacità di programmazione.
- **Application Composition** (1.27%): applicazioni troppo diversificate per essere gestite da soluzioni prepackaged ma che sono sufficientemente semplici da poter essere composte da componenti. Componenti tipici sono graphic user interface

(GUI) builders, database o object managers, middleware per processi distribuiti o processi transazionali, gestori hypermediali, smart data finders, componenti domain-specific come finanziari, medici, o controllo di processi industriali.

- **System Integration** (1.27%): sistemi a larga scala, sistemi ad alta integrazione o sistemi senza precedenti. Porzioni di questi software possono essere sviluppate tramite Application Composition ma generalmente richiedono una programmazione dedicata.

Il settore End-User Programming non necessita di COCOMO II come modello di stima in quanto questo tipo di applicazioni vengono generate in ore o giorni, pertanto una stima basata sull' attività è più che sufficiente. Il modello COCOMO II per Application Composition è basato sugli Object Points . La stima COCOMO II per i settori Application Generators, System Integration o Infrastructure è basata su una combinazione del modello per l' Application Composition (per gli sforzi di prototipazione rapida) e due modelli di stima incrementali per due successive porzioni del ciclo di vita. L ' appropriata sequenza di questi modelli dipenderà da fattori di mercato del progetto e dal suo grado di comprensibilità.

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

- **Application Composition:** modello che include gli sforzi di prototipazione per evitare elevati rischi potenziali come le interfacce utente, interazioni software/sistema, prestazioni, maturità tecnologica.
- **Early Design:** modello per la fase iniziale della progettazione (esplorazione alternativa di architetture software, di sistema e concetti operativi). A questo punto non si hanno sufficienti informazioni per ottenere una stima a granularità fine. Questo modello fa uso di Punti Funzione (FP - Function Points) e un ristretto numero addizionale di coefficienti di sforzo (cost drivers).

- **Post-Architecture:** modello per la fase di sviluppo ed il mantenimento del software. Fornisce stime precise se l ' architettura del ciclo di vita è stata ben definita rispetto gli obbiettivi del sistema, le operazioni concettuali ed i rischi. Questo modello ha la stessa granularità dei precedenti modelli COCOMO 81 e Ada COCOMO. Fa uso di linee codice sorgente (SLOC) e/o di Punti Funzione (FP) come metrica dimensionale, coefficienti per il riuso ed i guasti software (software breakage), 17 moltiplicatori di sforzo e 5 fattori che determinano l' esponente di scala del progetto (project ' s scaling exponent). Questi fattori sostituiscono i modi di sviluppo (Organic, Semidetached, Embedded) nel modello COCOMO 81 e perfezionano i quattro esponenti di scala di Ada COCOMO.

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

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

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

$$ESLOC = ASLOC \cdot \frac{AA + SU + 0{,}4 \cdot DM + 0{,}3 \cdot CM + 0{,}3 \cdot IM}{100}$$

dove:

- ESLOC – Equivalent SLOC: linee di codice equivalenti alle nuove istruzioni.
- ASLOC – Adapted SLOC: linee di codice di software da adattare

- AA – Assessment and Assimilation: valutazione ed integrazione. Il valore è ricavato dalla tabella 11.
- SU – Software Understanding: Comprensibilità del software. Il valore è ricavato dalla tabella 12.
- DM - Design Modification: percentuale di modifiche al progetto
- CM - Code Modification: percentuale di codice modificato
- IM – Integration Effort: percentuale di sforzo di integrazione originale necessario per integrare il software di riuso.

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

Scala per il calcolo dell'AA

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

Scala per il calcolo dell'SU

COCOMO II considera la volatilità dei requisiti attraverso il Moltiplicatore di Sforzo della Volatilità dei Requisiti (BRAK – Requirements Volatility effort multiplier).

Questo parametro viene utilizzato per correggere l' effettiva dimensione del prodotto ma non viene utilizzato nel modello Application Composition che incorpora già un meccanismo di correzione.

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

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

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

dove:

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

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

$$B = 1.01 + 0{,}01\sum_{i=1}^{5} Wi$$

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

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

Fattori di scala

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

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

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

Moltiplicatori di sforzo (EM-Cost Drivers)

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

$$T = \left[3.0E^{(0,33+0,2\cdot(B-1,01))}\right].\frac{SCEDPercentuale}{100}$$

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

Pro e Contro

L'introduzione del COCOMOII ha coperto alcuni aspetti mancanti a COCOMO, può utilizzare previsioni da Function Points ma è basato su variabili non ben definite quali gli Object Points.

COSMIC FFP

Il metodo COSMIC-FFP costituisce un metodo standardizzato di misurazione della dimensione funzionale del software nei domini funzionali comunemente denotati come 'software applicativo aziendale', o di tipo MIS (Management Information Systems – Sistemi Informativi Gestionali)', e 'software real-time'.

L'evoluzione

Un gruppo volontario di alcuni dei maggiori esperti nell ambito della misurazione del software a livello mondiale si incontrarono per la prima volta a Londra nel 1998 con l'obiettivo di '*sviluppare, testare, presentare al mercato e favorire l'accettazione di nuovi metodi di dimensionamento del software a supporto della stima e della misurazione delle performance'*.

Tali metodi dovevano permettere di misurare una dimensione funzionale del software applicativo gestionale (MIS), del software realtime/d'infrastruttura (OS e Middleware), a qualsiasi livello dell'architettura del software.

In seguito, il COSMIC, frutto delle esperienze degli ultimi 25 anni ottenute dalla Function Point Analysis IFPUG, e delle numerose proposte di miglioramento del metodo IFPUG, come la FPA MkII, la FPA NESMA e i Full Function Point, pubblicò l'insieme dei princìpi del metodo COSMIC FFP come il primo, vero Metodo di Misurazione della Dimensione Funzionale (FSM, Functional Size Measurement) di '2a generazione'.

Nel biennio 2000-2001 sono state svolte con esito positivo numerose 'prove

sul campo'. La più recente versione del metodo è stata pubblicata nel gennaio del 2003 (COSMIC-FFP 2.2).

All'inizio del 2003, COSMIC-FFP è divenuto uno Standard internazionale ISO/IEC 19761:2003.

Caratteristiche

Progettato per essere indipendente dalle decisioni implementative operate nella realizzazione del software da misurare, per essere applicabile a software applicativi aziendali a "prevalenza di dati" o "data rich", tipicamente richiesto in supporto alla gestione dei processi aziendali (software bancario, assicurativo, contabilità, il personale…), software real-time, e software ibridi (sia applicativi che real time), non è ancora progettato per prendere in considerazione la dimensione funzionale di porzioni di software, o loro parti caratterizzate da algoritmi matematici complessi o da altre regole specializzate e complesse o che elaborano variabili continue come suoni (audio) o immagini (video).

Fornisce uno strumento (il concetto di layer del software) per differenziare i requisiti funzionali utente allocati a differenti livelli di astrazione funzionale.

Rispetto ai metodi di misurazione della dimensione funzionale di '1^{a} generazione', il metodo COSMIC-FFP è applicabile a una gamma di tipi di software di gran lunga superiore agli altri metodi, si basa su concetti compatibili con i metodi di definizione dei requisiti e di progettazione del software, gli errori di applicazione sono meno probabili

e un dimensionamento automatico risulta possibile in alcune circostanze e con gli strumenti adeguati.

Misurare i 'requisiti utente funzionali' (FUR, Functional User Requirements) con il metodo COSMIC-FFP introduce dei benefici nel processo stesso di definizione dei requisiti:

- i requisiti misurabili sono con maggiore probabilità chiari e non ambigui.
- la scala di misura non è soggetta ad alcun limite di dimensione
- Le misure possono essere svolte da diversi ‘Punti di Vista della Misurazione’.

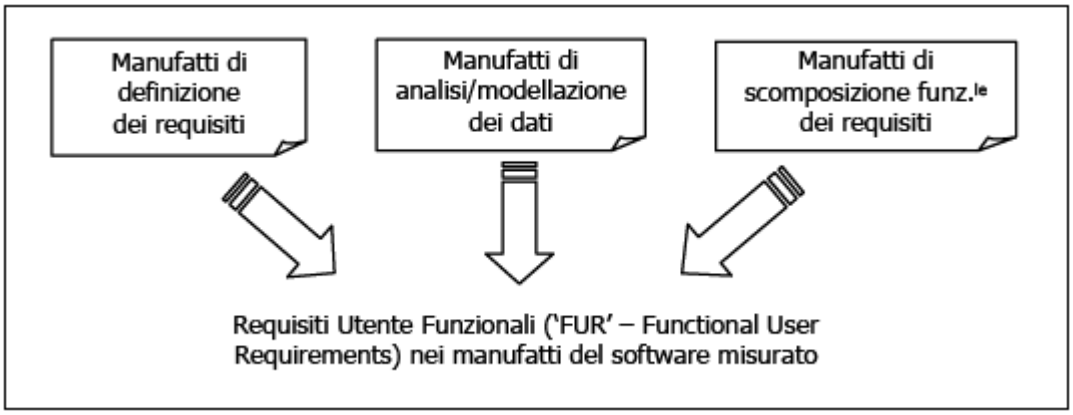
Dal Punto di Vista dello Sviluppatore, è possibile misurare distintamente le componenti dell’architettura software, modulare e stratificata in una data configurazione, richiesta per implementare i requisiti esaminati; oppure, il Punto di Vista dell’Utente Finale permette di misurare globalmente le funzionalità software richieste dall’utente applicativo, o dal committente, e eventualmente di confrontare la dimensione misurata con precedenti valori ottenuti con metodi di ‘1ª generazione’.

Il metodo

Il metodo di misurazione COSMIC-FFP richiede l'applicazione di un insieme di regole e procedure ad una data porzione di software così come essa è percepita dal punto di vista dei suoi Requisiti Funzionali Utente FUR. Il risultato dell'applicazione di queste regole e procedure è un valore numerico di una quantità che rappresenta la dimensione funzionale del software.

ESTRAZIONE DEI REQUISITI UTENTE FUNZIONALI

Le funzionalità fornite dal software agli utenti (FUR) indicano ‘cosa’ il software deve fare per gli utenti e escludono ogni altro requisito tecnico o di qualità che dica ‘come’ il software deve operare.



I FUR sono costanti, poiché, indipendentemente dal tipo di manufatto dell'ingegneria del software da cui sono estratti, i FUR risultanti esprimono in ogni caso solo la definizione delle funzionalità fornite dal software ai propri utenti.

FASE DI MAPPATURA

La fase di mappatura necessita di due modelli. Il primo modello, il 'modello del contesto del software', riconosce che il software oggetto di misurazione può esistere in un contesto di hardware e/o di altri strati software e che sarà separato dai propri utenti da un confine. Lo stesso software può consistere di vari strati e di elementi di pari livello all'interno degli strati. Occorre una comprensione condivisa di questi concetti e di come essi sono correlati l'uno all'altro. Il secondo modello è il 'modello del software generico' COSMIC-FFP in cui i FUR devono essere espressi prima di applicare il processo di misurazione.

È importante notare che questi due modelli sono validi strumenti per l'espressione dei FUR, anche nel caso in cui non vi sia alcuna intenzione di effettuare una qualsiasi misurazione.

Prima del modello del software generico COSMIC-FFP, occorre descrivere il modello del contesto del software.

Modello del Contesto

Un aspetto chiave della misurazione della dimensione funzionale del software è la definizione di cosa è considerato parte del software e cosa è considerato parte dell'ambiente operativo del software.

Il software è delimitato dall'hardware di I/O come mouse, tastiere, stampanti e schermi, o da dispositivi vari come sensori e relè. In direzione del cosiddetto 'back-end', il software è delimitato dall'hardware di memoria persistente come dischi rigidi e memorie RAM e ROM. Il flusso funzionale degli attributi dei dati può essere caratterizzato da quattro distinti tipi di movimento, in direzione del 'front-end', due tipi di movimento (Entry e eXit) consentono lo scambio di dati con gli utenti attraverso un 'confine'. In direzione del 'back-end', due tipi di movimento (Read e Write) consentono lo scambio degli attributi dei dati con l'hardware della memoria persistente.

Modello del Software Generico

Il modello del software generico COSMIC-FFP assume che siano soddisfatti due principi generali:

- Principio Generale 1: il software che deve essere mappato e misurato è alimentato da input e produce output utili, o un risultato utile, per gli utenti.
- Principio Generale 2: il software che deve essere mappato e misurato manipola porzioni di informazione, denotate come 'gruppi di dati', che consistono di attributi dei dati.

Il modello del software generico COSMIC-FFP distingue quattro (tipi di) sotto-processi di movimento dei dati:

- Entry: muovono dati dagli utenti attraverso il confine verso il processo funzionale

- eXit: muovono dati dal processo funzionale attraverso il confine verso gli utenti
- Read e Write: muovono dati da e verso la memoria persistente.

FASE DI MISURAZIONE COSMIC-FFP

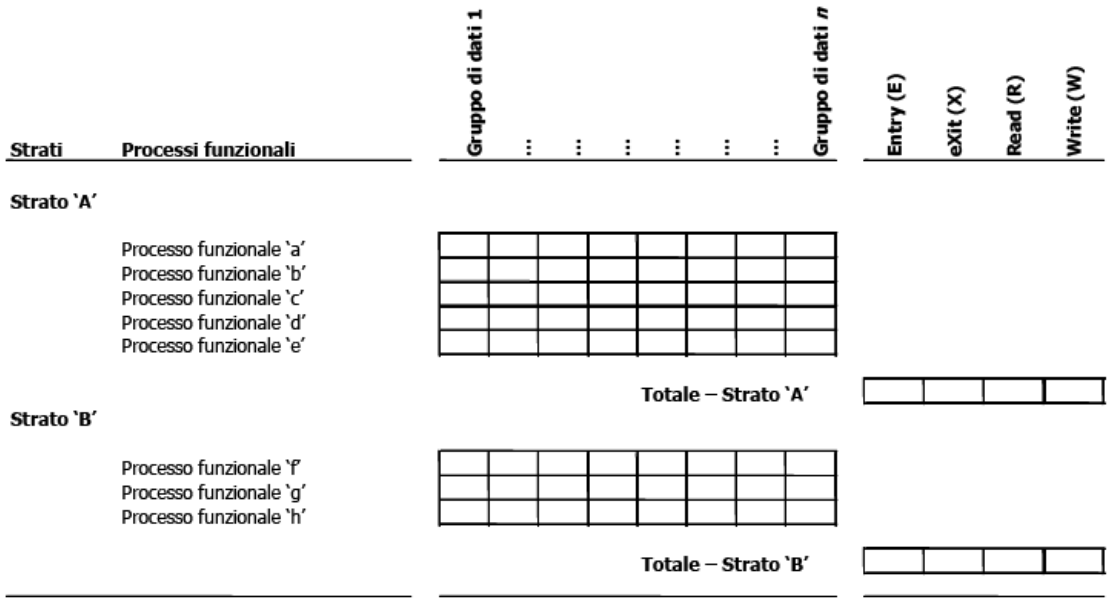
La fase di misurazione COSMIC-FFP prende in input un'istanza del modello del software generico e, mediante un insieme definito di regole e procedure, produce in output un valore di una quantità, la cui grandezza è direttamente proporzionale alla dimensione funzionale del modello, in base al seguente principio.

La dimensione funzionale del software è direttamente proporzionale al numero dei suoi movimenti di dati. Tra le caratteristiche dell'insieme di regole e procedure che guidano la produzione di questo valore di una quantità vi sono le seguenti:

1. *Funzione di Misurazione*: ad ogni occorrenza di movimento di dati è assegnata una quantità numerica, in base al suo tipo, tramite una funzione di misurazione.
2. *Unità di Misura: l*o standard di misurazione, 1 cfsu (Cosmic Functional Size Unit), è definito equivalente per convenzione a un singolo movimento di dati.
3. *Proprietà Additiva*: la dimensione funzionale di un processo funzionale è definita come la somma aritmetica dei valori della funzione di misurazione, applicata a ciascuno dei suoi movimenti di dati. La dimensione funzionale di una qualsiasi modifica funzionale richiesta per una porzione di software è per convenzione la somma aritmetica delle dimensioni funzionali di tutti i movimenti di dati aggiunti, modificati e cancellati in quella porzione di software.
4. *Sotto-unità di Misura*: qualora si richieda una maggiore precisione nella misurazione dei movimenti di dati, è possibile definire una sotto-unità di misura. Il più piccolo processo funzionale deve avere almeno un Entry e un eXit o un Write.

RIEPILOGO

Il metodo COSMIC FFP è riassumibile nel seguente schema di raccolta dati, per registrare tutte le componenti identificate del modello del software generico oggetto di misurazione:



FASE DI MAPPATURA

- Ogni gruppo di dati identificato è registrato in una colonna.
- Ogni processo funzionale è registrato in una specifica riga, raggruppato per strato identificato.

FASE DI MISURAZIONE

- Per ogni processo funzionale identificato, i movimenti di dati identificati sono riportati nella casella corrispondente con la seguente convenzione: ‘E’ per un Entry, ‘X’ per un eXit, ‘R’ per un Read e ‘W’ per un Write;

- Per ogni processo funzionale identificato, i movimenti di dati sono successivamente sommati per tipo e ogni totale è registrato nella colonna appropriata all'estrema destra della matrice;
- Si può quindi calcolare e registrare la sintesi della misurazione per ogni strato, nelle caselle con bordi, sulla riga di 'Totale'.

Pro e Contro

Oltre ad essere applicabile in ambienti di controllo real-time, sono molto più semplici da calcolare rispetto agli IFPUG Function Points.

Analisi delle metriche del software

Lo studio delle metriche del software ha portato alla formulazione di molti quesiti: volevamo comprendere meglio quale era il livello di accettazione di ogni metodo e quali erano le aspettative di chi il software lo deve misurare per utilizzare.

Dopo molti incontri con la comunità del Software Metrics, con il gruppo di sviluppo Open Source di JavaOpenBusiness e numerose discussioni con esperti del settore, si è giunti alla formulazione di un questionario che è stato utile a comprendere il punto di vista della comunità degli utilizzatori delle metriche.

Nella stesura del questionario sono stati tenuti in considerazione alcuni aspetti fondamentali quali le dimensioni dell'azienda, il tempo medio impiegato per la misurazione del software, la conoscenza e la valutazione delle metriche più comuni e una valutazione per ognuna di essa oltre, infine, ad una valutazione soggettiva di quelli che sono i bisogni degli utilizzatori nell'ambito della misurazione del software.

Il questionario è stato distribuito su tre diverse liste di distribuzione, una italiana e due americane.

Sono stati ottenuti risultati più che soddisfacenti ottenendo un totale di 22 risposte. Il bacino di utenza riscontrato è omogeneo, si nota una provenienza da tutti i settori industriali. Si è verificata però, la presenza di industrie molto grandi, oltre il 60% oltre i mille dipendenti, mentre solo l'8% di Piccole Medie Imprese.

La quasi totalità degli intervistati impiega per la stima di un progetto software oltre i due giorni basandosi principalmente sull'esperienza personale.

I risultati completi del sondaggio sono riportati in appendice.

Capitolo

4

METRICHE OPEN SOURCE

Un confronto fra software Open Source e proprietari è viziato dall'impossibilità di accedere al codice sorgente degli applicativi "closed source". Esistono alcuni modelli di confronto basati sulle funzionalità, caratteristiche tecniche e prestazioni. Dato che i rispettivi modelli di sviluppo sono differenti, i parametri da tenere in considerazione sono completamente diversi, sarebbe assurdo valutare l'attività di rilascio per un software proprietario, quando gli sviluppatori sono legati ad esigenze commerciali.

L'attività di valutazione di un prodotto Open Source è ancora in fase di studio, la scelta di un prodotto maturo, risulta essere ancora affidata alla sola esperienza dei manager. A tal fine, negli ultimi tre anni, sono stati introdotti quattro modelli di valutazione.

Il primo modello storico di valutazione, è quello del costo di possesso, utilizzabile anche in sovrapposizione ai modelli seguenti, che fornisce unicamente un indicazione del solo costo relativo al possesso di un prodotto.

Nel 2003 è stato introdotto da Capgemini[17] il primo modello di valutazione per prodotti Open Source: l'Open Source Maturity Model, successivamente modificato e integrato da Navica[18]. Nel 2004 Atos Origin[19], ha pubblicato il modello QSOS (Qualification and selection of Open Source Software). Infine nel 2005 è stato introdotto, da un consorzio di aziende ed università americane, l'Open BRR.

Tutti i modelli sono basati sulla valutazione soggettiva di un certo numero di parametri, quindi dell'attribuzione di un fattore peso in relazione all'importanza del parametro. La

valutazione ottenuta, sarà quindi personale. Il punteggio finale che ogni metodo restituirà non sarà “universale” bensì utile solo ad aziende di tipo omogeneo.

Costo di possesso (TCO)

In tempi recenti alcune software house, hanno dato incarico a note società di consulenza internazionale di dimostrare il minor costo totale di possesso (TCO=Total Cost of Ownership) dei programmi commerciali rispetto a quelli non commerciali; altro indice a corredo è il Ritorno sull'Investimento (ROI = Return On Investment) ed i Quadrati Magici.
Il TCO ha l'obiettivo di stimare per un lungo periodo la somma dei costi, visibili e nascosti, interni ed esterni, dei prodotti e servizi di informatica mentre i Quadranti Magici hanno lo scopo di visualizzare graficamente il posizionamento dei prodotti/servizi in relazione alle loro capacità di visione verso il futuro e della sua concreta realizzazione.

I fautori dei programmi commerciali presentano tabelle con TCO, ROI e Quadrati Magici positivi per i programmi commerciali e negativi per il software libero ed aperto; esattamente al contrario di quello che normalmente avviene da parte dei fautori del software libero ed aperto.

Il TCO è fondamentalmente un'analisi statica dei costi di esercizio di una apparecchiatura. Non tenendo conto di eventuali ritorni, è applicabile solamente durante la fase di analisi strategica degli investimenti, inoltre il TCO presenta dei costi connessi molto onerosi il che lo proietta solamente nel campo applicativo di grandi aziende. [5]

La teoria su cui è basato il TCO pone in rilievo la dipendenza del costo totale di utilizzo di una apparecchiatura IT non solo dai costi di acquisto ma anche da altri fattori legati al suo ciclo di vita.

Il TCO si presenta come un metodo ottimo per misurare i costi totali, identificando tutte le spese, in termini chiari e facilmente misurabili; ma presenta lo svantaggio di non

poter stimarne il valore aziendale complessivo. Per la sua estrema semplicità e linearità nel fornire un primo quadro della situazione, solitamente viene utilizzato come unico strumento di analisi tralasciando altri sistemi più importanti e senza valorizzare altri tipi di benefici più difficili da misurare e più controversi[13]. L'analisi TCO deve quindi tener conto di:

- costi per l'acquisto dei componenti hardware o software (ricerca del fornitore sul mercato, costi di amministrazione per le ricerche di mercato, costi delle licenze software);
- costi per lo sviluppo di personalizzazione degli applicativi implementati dai dipendenti interni;
- costi operativi,legati all'aggiornamento e alla manutenzione e all'esercizio del software; questi costi denominati "costi operativi" comprendono: formazione di personale IT e di end-users, supporto degli end-users nei problemi riscontrati nell'utilizzo della tecnologia, gestione della sicurezza informatica, utilizzo di spazi per ospitare apparecchiature hardware (es. server, mainframe), consumi di energia, costi di connessione Internet, costi derivanti dal down-time del sistema per malfunzionamenti o errori degli end-users;
- costi legati alla dismissione del sistema (smantellamento delle apparecchiature hardware, eliminazione dei cavi portanti delle reti LAN).

Il TCO rappresenta solo una parte dei blocchi funzionali che rientrano nel Full Business Value (FBV). Il Full Business Value ovvero l'intero valore di un investimento aziendale si compone sostanzialmente di quattro componenti:

- efficienza dei sistemi,
- efficacia dei sistemi,
- efficienza del business

- efficacia del business

Il TCO pone maggiore interesse per l'efficienza dei sistemi, coprendo solo una parte dell'intera analisi[13].

Il TCO deve il suo largo utilizzo alla sua peculiarità di marcare maggiormente costi puramente di natura IT trascurati per decenni[13].

In letteratura è possibile trovare molti esempi su come il TCO può rilevare importanti benefici, ma fallire nel descrivere il valore totale. Secondo uno di questi si può supporre di voler motivare una richiesta di upgrade per un server e una rete, in una situazione di forte pressione affinché l'IT riduca le sue spese. Avvalendosi del metodo TCO si identificano tutti i costi dell' intero ciclo di vita (acquisto, installazione, gestione, supporto e manutenzione). A questo punto, se la scelta di mantenere la situazione attuale costa praticamente zero per l'acquisto, ma costa molto di più per l'esercizio rispetto alla scelta di utilizzare nuovi componenti, l'analisi evidenzierà quantitativamente questi fattori. Così pure il TCO catturerà elementi come il costo e il rischio di interruzioni del servizio, i costi di manutenzione e di supporto. Fin qui è tutto perfetto e si ottiene un'analisi dei costi difendibile. Ma qual è il suo valore aziendale totale? Per ottenere questo valore bisogna aggiungere gli altri tre componenti descritti sopra del Full Business Value. Per analizzare l'efficacia dei sistemi si possono quantificare fattori come la capacità di portare in esercizio applicazioni interamente nuove come il CRM, per l'efficienza del business si potrebbe inglobare il valore di una riduzione sulle spese di viaggi a causa del miglior supporto fornito dal CRM, da un server potenziato e da un'infrastruttura di rete. Per quanto riguarda l'efficacia del business si possono prendere in considerazione i valori evidenti e non nel possedere un settore vendite in grado di introdurre più nuovi prodotti più rapidamente, sempre a fronte dell' introduzione del CRM. Utilizzando il Full Business Value, unitamente al TCO,oltre ad identificare i benefici addizionali a supporto del nostro business case, si sarebbe in grado di incrementare la credibilità dell'IT dimostrando ai dirigenti degli altri settori la propria conoscenza di tutto il sistema aziendale"[14].

Giacomo Cosenza di Sinapsi propone un altro indice molto interessante poiché consente di misurare qualsiasi tipo di programma od applicazione, con riferimento a quanta libertà è lasciata all'utente finale.

L'indice ha l'acronimo di TAO (Total Account Ownership) e può essere tradotto in Possesso Totale del Cliente nel senso di dipendenza dei clienti dal fornitore di tecnologia.
Un valore prossimo allo zero esprime un basso livello di dipendenza, mentre un livello prossimo all'uno esprime, evidentemente, un alto livello di dipendenza.

"Si noti che il TAO-Index ha per complemento quello che chiamerei Liberty-Index (indice di libertà), se non fosse che l'uso del temine libertà nei contesti tecnologici rischia di apparire presuntuoso se non addirittura offensivo per tutti coloro che dell'assenza di libertà, nel senso più ampio e umano del termine, hanno sofferto in passato o soffrono tutt'oggi".

Se, ad esempio, gli elementi presi in considerazione nella determinazione dell'indice di dipendenza (TAO) di un programma proprietario fossero: dipendenza contrattuale, licenze di uso punitive per l'utilizzatore finale, dipendenza di uso e chiusura all'innovazione, impossibilità di mutare le logiche elaborative del programma, dipendenza da formati proprietari degli archivi di collegamento, obbligo di sottoscrivere contratti di assistenza e di manutenzione, dipendenza da hardware particolare, da versioni sempre aggiornate del Sistema Operativo o da altri programmi proprietari, dipendenza da consulenti certificati dal produttore, aggiornamenti automatici invasivi del proprio computer, documentazione carente dal punto di vista tecnico, e ad ogni dipendenza indicata fosse dato un peso unitario di 0.1, sarebbe molto probabile ottenere un TAO prossimo all'unità con un Indice di Libertà prossimo allo zero.

Alle PMI potrebbe essere conveniente tenere conto, sia degli elementi economici dell'indice del Costo Totale di Possesso (TCO), sia del valore dell'Indice di Possesso Totale del Cliente (TAO).

Molto dipende dall'importanza che è data alla facilità di liberarsi di un fornitore di programmi proprietari in relazione all'importanza che si dà ai soli aspetti economici; in relazione ai reciproci fattori di importanza, il TCO ed il TAO saranno valorizzati con differenti pesi per formare l'indice globale del TCAO.

E' molto importate evidenziare che, mentre il TCO può essere calcolato dal mega consulente galattico dell'ICT o dallo specialista in Informatica, il TAO ed i pesi dei contributi di valorizzazione del TCO e del TAO per la formazione dell'indice globale del TCAO, sono prerogativa del Cliente finale e dei suoi atteggiamenti in relazione alla conservazione delle libertà di poter cambiare, evitando la prigione a vita della tecnologia proprietaria.

Il Cliente finale, in questo modo, ritorna ad essere arbitro e libero valutatore dei programmi e delle applicazioni da installare nel suo computer.

Open Source Maturity Model

Ogni giorno vengono introdotti sul mercato nuovi prodotti incompleti, con funzionalità limitate e un alto numero di bugs. Successivamente, i produttori, attraverso nuovi aggiornamenti forniscono al prodotto le funzionalità aggiuntive mancanti e lo rendono più stabile.

Il concetto di maturità del software è riconosciuto universalmente, ma quando è possibile considerare un prodotto maturo?

Un prodotto può essere considerato maturo quando possiede le seguenti caratteristiche:

- Completezza
- Livello di qualità elevato
- Longevità
- Aggiornabilità
- Robustezza

La maturità di un prodotto è la chiave per capire il grado di soddisfazione di un prodotto. I prodotti non maturi possono essere usati per i sistemi non critici, mentre l'uso in fase di produzione richiede che i prodotti siano molto maturi. Molti progetti falliscono quando il sistema non esegue le funzionalità previste o dimostrano prestazioni poco efficienti. D'altro canto i costi per intervenire in un sistema "fallato" sono molteplici con conseguente perdita di tempo utile sul mercato.

A tal scopo, è stato introdotto l'Open Source Maturità Model (OSMM), uno standard aperto che dovrebbe facilitare la valutazione e l'adozione dei software Open Source.

. La valutazione di un software che basa il proprio metodo sull'Open Source Maturity Model assume che *la validità e la solidità di un prodotto Open Source siano direttamente proporzionali alla maturità del progetto*: un software è considerato maturo quando il team è attento alle funzionalità del prodotto ma anche quando progetta una rigorosa attività di rilascio e di supporto.

L'utilizzo si basa sulla valutazione di ogni elemento di prodotto assegnando una priorità per ogni elemento. Successivamente si potrà calcolare il punteggio di maturità finale attraverso i pesi assegnati ad ogni elemento.

	Phase 1: Assess Element Maturity				Phase 2	Phase 3
	Define Requirements	Locate Resources	Assess Element Maturity	Assign Element Score	Assign Weighting Factor	Calculate Product Maturity Score
Product Software						
Support						
Documentation						
Training						
Product Integrations						
Professional Services						

Il metodo si compone di tre fasi distinte:

1. Valutazione della maturità degli elementi
2. Assegnamento di un fattore peso
3. Calcolo del punteggio di maturità finale

Fase 1: valutazione della maturità di prodotto:

La prima fase identifica i fattori chiave per la valutazione del livello di maturità di ogni elemento, si considerano elementi chiave:

- Prodotto Software
- Documentazione
- Supporto
- Training
- Integrazione di prodotto
- Servizi professionali

Ad ogni elemento viene assegnato un punteggio valutando quattro fasi:

- Definizione dei requisiti
- Locazione delle risorse
- Valutazione della maturità (da non esistente a prodotto pronto per la produzione)
- Assegnamento del punteggio di maturità

Fase 2: assegnamento di un fattore peso

In base alle proprie esigenze, si assegna un peso ad ogni punteggio. I pesi consentono ad ogni elemento di riflettere la propria importanza in relazione alla maturità dal prodotto.

I valori di default per i pesi sono:

Software:	4
Support:	2
Documentation:	1
Training:	1
Integration:	1
Proferssional Services:	1

Fase 3: calcolo del punteggio di maturità finale

Dopo aver valutato e assegnato un fattore peso ad ogni elemento, il punteggio totale di maturità sarà dato dalla somma di tuti i valori precedentemente calcolati.

Considerazioni

Attraverso l'OSMM è possibile valutare velocemente prodotti O.S. determinandone il livello di maturità.

Durante la fase di scelta di un prodotto, l'OSMM può rivelarsi uno strumento molto veloce per determinare il livello di maturità al fine di ottenere un quadro completo della situazione attuale.

Di contro, non è possibile avere una misurazione univoca ed oggettiva.

Come per ogni strumento, può essere usato più o meno coscientemente; in ogni caso bisogna sempre fare attenzione a non considerarlo un "silver bullet".

Applicato in modo scorretto potrebbe portare a pessime decisioni molto velocemente, applicato invece "con coscienza" può ritenersi un valido assistente a supporto delle organizzazioni IT al fine di effettuare la giusta scelta in modo efficiente.

Open Business Readiness Ratings

Intel, in collaborazione con un'università americana il Center for Open Source Investigation (COSI) della Carnegie Mellon University e SpikeSource, ha avviato un progetto per la creazione di un sistema di valutazione standard del software Open Source con lo scopo di fornire agli utilizzatori professionali un indice attendibile di maturità delle oltre 100mila iniziative di sviluppo Open Source oggi censite su SourceForge, CodeHaus, Tigris, Java.net e Open Symphony.

Nelle intenzioni dei proponenti, il modello sarà adattabile per riflettere le differenti tipologie di utilizzo: per esempio, distinguendo tra i requisiti delle università e quelli, molto differenti, delle grandi aziende.

Sul sito ufficiale www.openbrr.org è possibile trovare svariati esempi di applicazione del modello Open BRR, utili ad una rapida valutazione personale del prodotto. Essendo basati su semplici fogli di calcolo (excel e openoffice), è inoltre possibile variando semplicemente alcuni parametri di valutazione personale, adattare la stima alle proprie esigenze. Dal sito ufficiale risulta anche che al momento il progetto è in fase di test della durata di un anno, quindi inizierà ad essere indicato in tutti i progetti Sourceforge e Java.net

"*Il modello che stiamo creando permetterà a utenti e sviluppatori di capire se un determinato software è appropriato o meno per le proprie necessità, consentirà alle aziende utenti di ridurre il tempo che devono dedicare alla valutazione in proprio della qualità del software e della continuità dei progetti Open Source. BRR consentirà a tutti, utenti e sviluppatori, di dare il proprio contributo in modo che sia utile agli altri*".[1]

Il BRR si compone di quattro fasi: prima si decide la lista di software candidati, quindi si ordinano le categorie per importanza, quindi si processano i dati ed infine, si traducono i dati nel Business Readiness Rating.

[1] http://www.cwi.it/showPage.php?template=articoli&id=13550

Fase 1: quick assessment:

Si identifica una lista di componenti da valutare

Si valuta ogni componente tenendo conto di alcuni indicatori, assegnando un punteggio da 1 a 5 dove uno indica “inaccettabile” e 5 “Eccellente”.

L’obiettivo è quello di eliminare i prodotti che sono chiaramente inadatti.

Gli indicatori da considerare sono:

1. Il target di utilizzo dell’applicazione (mission-critical, Regular, Development, Experimentation)
2. Selezionare un elenco di indicatori basati sul target di utilizzo:
    - Tipo di licenza:
        - Licenze approvate dall’Open Source Initiative
        - Copyleft License

      Il tipo di licenza può essere importante per l’effettivo bisogno che si ha del software, se ad esempio si vuole modificare un prodotto e poi rivenderlo a proprio nome, bisogna verificarne la fattibilità nella licenza.
    - Rispetto degli standard

- Esistenza di utenti referenziabili che utilizzano il componente: è importante sapere chi utilizza il prodotto e che tipo di utilizzo ne fa
- Supporto fornito da un'organizzazione stabile.
- Linguaggio di implementazione

  Il linguaggio potrebbe essere importante nel caso di aziende con un'ottima padronanza di un tale linguaggio e mancanza di competenza su di un altro.
- Supporto per l'internazionalizzazione o per la lingua desiderata.
- Presenza di moduli aggiuntivi sviluppati da terze parti.
- Presenza di libri pubblicati sul prodotto
- Il prodotto è seguito da analisti, tipo Gartner o IDC?

3. Aggiungere altri indicatori alla lista se necessario: alcuni tipi di prodotti potrebbero necessitare di criteri di valutazione ulteriori.
4. Creazione di regole di accettazione degli indicatori specificati
5. Valutazione di tutti i componenti

Fase 2: Target usage assessment

La seconda fase nell'ordinamento dei prodotti, dal migliore al peggiore, rispetto al punteggio calcolato nella fase 1.

Fase 3: Data collection and processing

La terza fase consiste nella raccolta dei dati, indicando tutti i punteggi in un scala da uno a cinque.

Quindi si passa alla verifica delle funzionalità di prodotto. La verifica delle funzionalità è una valutazione computazionalmente diversa dalle altre categorie. Ogni tipo di applicazione ha un insieme diverso di caratteristiche.

Il punteggio relativo alle funzionalità è ottenuto inizialmente confrontando le caratteristiche del componente rispetto ad un “uso standard” . Una volta disponibile l’insieme di caratteristiche standard, è possibile iniziare la valutazione delle caratteristiche attraverso quattro fasi:

1. Assegnare un punteggio di importanza a tutti gli elementi della lista delle caratteristiche, utilizzando una scala da 1 a 3, dove 1 è poco importante e 3 molto importante
2. Comparazione della lista delle caratteristiche del componente con le caratteristiche standard. Per ogni caratteristica, sommare il punteggio di importanza, se non è implementata una caratteristica standard, sottrarre il punteggio.
3. Se il software ha delle caratteristiche ulteriori non incluse in quelle standard, assegnare un punteggio di importanza ad ogni funzionalità e sommarla al punteggio di importanza.
4. Calcolare il punteggio delle caratteristiche dividendo il punteggio di importanza totale con il massimo ottenibile dalla somma dei punteggi delle caratteristiche standard. Con tale punteggio è possibile sapere la percentuale di caratteristiche implementate dal prodotto.

Fase 4: Data translation

L'ultima fase consiste nella normalizzazione del punteggio delle caratteristiche, su una scala da 1 a 5 nel seguente modo:

- Meno del 65% ➔ 1 (inaccettabile)
- 65% - 80% ➔ 2 (quasi accettabile)
- 80% - 90% ➔ 3 (accettabile)
- 90% - 96% ➔ 4(ottimo)
- Più del 96% ➔5 (eccellente)

Considerazioni

L'Open BRR introduce numerosi criteri di valutazione, totalmente personalizzabili, in base ad una semplice analisi dei requisiti.

Al momento non si hanno notizie chiare di come sarà calcolato l'indice su tutti i prodotti Sourceforge, anche se è presumibile che i valori di importanza delle caratteristiche verranno indicati come uguali.

In una possibile implementazione, si ritiene molto utile la creazione di una tabella in cui sia possibile variare i pesi secondo la propria esigenza, variando l'indice per ogni prodotto.

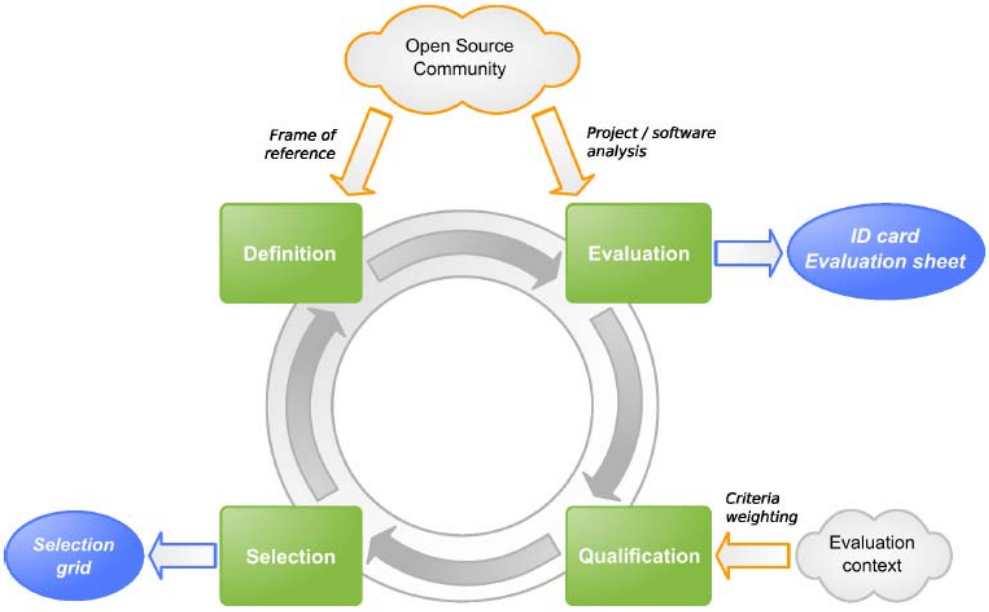
L'indice se ben implementato potrà essere un ottimo modo di ricerca di prodotti O.S. se implementato a ritroso, supponendo di cercare un certo tipo di applicativo, sarebbe molto utile poter indicare i propri pesi di importanza in modo da avere quindi l'elenco dei prodotti ordinati per Open BRR.

Qualification and Selection of Open Source Software

QSOS è un metodo rivolto a selezionare e comparare prodotti free e Open Source. Proposto nel 2004 da Atos Origin, sotto licenza GNU, si pone quale valida alternativa all'Open BRR e all'OSMM.

Il processo si compone di quattro step indipendenti ed iterativi:

- *Definition:* Definizione di una struttura di riferimento, da utilizzare nei passi successivi (licenza, comunità, griglia funzionale…).
- *Evaluation:* Valutazione del software su tre assi principali: functional coverage, rischi per l'utente, rischi per il fornitore dei servizi (expertise, training, support services). Ogni asse contiene più criteri.
  Per esempio, l'Analisi dei rischi utente include la curabilità intrinseca, l'adattabilità tecnica, l'industrializzazione e la strategia.
- *Qualification:* qualificazione di uno specifico contesto dell'utente (company o individual) per la pesatura dei criteri precedenti
- *Selection:* selezione e comparazione dei prodotti software



Fase 1: Defintion

Obiettivo di tale fase è quello di definire vari elementi riutilizzati dai tre step successivi .

Gli elementi da valutare sono:

- Software families
- Tipo di licenza
    - Proprietà
    - Viralità
    - Ereditarietà
- Tipo di comunità
    - Sviluppo isolato
    - Gruppi di sviluppatori
    - Organizzazioni di sviluppatori
    - Società commerciali

Step 2: Evaluation

L'obiettivo è quello di collezionare tutte le informazioni della comunità Open Source.

Per ogni prodotto si raccoglie un insieme di informazioni *identity card (ic)* software dove vengono indicati i seguenti fattori:

- Informazioni generali:
    - Nome del software
    - Riferimenti, data di creazione, data di rilascio della id card
    - Autore

    - Tipo di software
    - Breve descrizione
    - Tipo di licenza
    - Link al progetto (url)
    - Sistemi operativi compatibili
    - Fork’s origin
- Servizi esistenti
    - Documentazione
    - Numero di offerte di supporto commerciali
    - Numero di offerte di training
    - Numero di consulenti
- Aspetti tecnici e funzionali
    - Tecnologia di implementazione
    - Prerequisiti tecnici
    - Funzionalità dettagliate
    - Roadmap
- Sintesi
    - General trend
    - Commenti

Dopo la creazione delle identità card, si descrive ogni release del prodotto in un foglio di valutazione. Tale documento include informazioni molto più dettagliate delle identità card, andando a specificare tutti gli aspetti che verranno ritenuti critici.

Si assegna quindi un punteggio da 0 a 2 a tutti i criteri. Tale punteggio, sarà poi utilizzato nello step quattro.

Si rimanda la descrizione di tutti i parametri indicati alla documentazione ufficiale, il cui estratto è reperibile in appendice C.

Fase 3: qualificazione

Durante questa fase si devono definire dei requisiti che riflettano i bisogni e i vincoli relativi alla selezione dei prodotti.

Un primo livello di selezione, può essere definito sui dati delle id card.

Per esempio può essere definito un vincolo relativo al sistema operativo su cui il prodotto deve funzionare.

In generale, i requisiti non sono mandatari e non includono alcun fattore di peso, il loro utilizzo serve solo per l'eliminazione dei fattori inadeguati raccolti nelle id card.

Fase 4: selezione

L'ultima fase consiste nella selezione del prodotto che meglio risponde alle esigenze dell'utilizzatore.

Sono possibili due diversi tipi di selezione:

- Strict selection
- Loose selection

Strict selection

La selezione è fortemente restrittiva, si basa direttamente sull'eliminazione diretta rimuovendo automaticamente i prodotti con identity card non compatibili con i requisiti specificati nella fase 3 non appena una caratteristica di prodotto non risponde completamente ai requisiti formulati.

Loose selection

Questo metodo è meno rigido del precedente, invece di eliminare direttamente i prodotti non eleggibili, crea una classifica assegnando un gaps ai requisiti non pienamente soddisfatti.

La pesatura delle funzionalità è basata sul livello di requisito definito nella fase precedente sotto forma di griglia funzionale:

Livello di requisito	Peso
Funzionalità richiesta	**+3**
Funzionalità opzionale	**+1**
Funzionalità non richiesta	**0**

Considerazioni

Dal punto di vista operativo, questo modello di valutazione, non presenta differenze rilevanti rispetto al panorama esistente.

Si nota una certa incongruenza nel metodo: nella seconda fase, si richiede un notevole sforzo nella definizione di ogni fattore delle identity card indipendentemente dal peso che verrà attribuito singolarmente. Nelle fasi successive si chiede di eliminarli in base al peso a loro assegnato.

A fronte di una piccola modifica al metodo, sarebbe possibile ottenere una discreta riduzione dei tempi di stima.

Nel metodo attuale, si valutano inizialmente tutti i parametri definiti nelle identity card, a cui viene poi attribuito un fattore peso con cui calcolare infine l'influenza di tale fattore.

Supponendo di avere fattori con incidenza zero, cosa peraltro molto frequente, si verifica un notevole dispendio di tempo nella valutazione di tali fattori che, durante la quarta fase, verranno moltiplicato per zero e quindi eliminati.

Si ritiene che, un semplice scambio delle fasi potrebbe diminuire notevolmente il tempo di stima eliminando direttamente tutti i fattori con peso zero, prima della compilazione di ogni identity card. In tal modo, si risparmieranno automaticamente i tempi di valutazione di tutte le funzionalità non rilevanti.

Sotto l'aspetto funzionale, ritengo invece il metodo assolutamente equivalente all'Open BRR e all'OSMM. Presenta una miglior definizione di alcuni requisiti (studio della comunità, del supporto fornito) ma risulta carente rispetto all'Open BRR sotto l'aspetto delle qualità interne; non viene assolutamente menzionata la possibilità di una valutazione delle qualità interne di prodotto.

Conclusioni

La valutazione delle funzionalità, è il cuore della stima del software.

Tutti i metodi trattati consentono qualche forma di valutazione, utile per l'adozione di un nuovo prodotto O.S,

Una comparazione dei modelli, può essere utile per meglio comprendere le differenze tra ognuno di essi.

Il punteggio finale ottenuto attraverso i modelli indicati, si basa sull'assegnazione soggettiva di un fattore peso.

Ogni modello fornisce una valutazione finale normalizzata, le differenze tra l'applicazione dei diversi modelli si basa sulla definizione più o meno dettagliata di alcune caratteristiche di prodotto da valutare.

A differenza dell'OSMM, l'Open BRR e il QSOS possono essere applicati iterativamente.

Similarmente, i modelli definiscono un'area di valutazione, il BRR definisce 12 aree di valutazione consigliandone l'utilizzo di almeno sette, permettendo però la possibilità di inserire ulteriori aree. L'OSMM indica solo sei aree di valutazione rigide e non estendibili.

Si deduce che, in un modello di valutazione flessibile, le capacità di valutazione siano molto ridotte.

Capitolo

5

OPEN BUSINESS QUALITY RATING:
una proposta alternativa

Il software Open Source si è affermato in questi anni come valida alternativa a quello proprietario.

Di contro, è ancora molto difficile individuare e scegliere, tra le tante proposte esistenti, i prodotti che meglio si adattano alle specifiche esigenze.

Attraverso un contatto più ravvicinato con il mondo Open Source e in particolare con gli sviluppatori e gli utilizzatori di programmi Open Source, abbiamo successivamente potuto apprezzare in prima persona altri aspetti molto importanti:

- i prodotti O.S. nascono e crescono, quantitativamente e qualitativamente, su base collaborativa, che fa dell'espansione e dell'adattamento alle diverse situazioni un principio fondante: i modelli operativi non sono pertanto imposti, ma proposti, e ciascun soggetto, se ne è capace, può ridefinirli e rimetterli in circolo;
- i diversi programmi e le diverse distribuzioni si caratterizzano spesso perché sono il risultato della fatica di autentici "addetti ai lavori" di specifici settori, cioè di persone che hanno deciso di costruire (o di adattare) un ambiente digitale in modo da avere risposte il più possibile soddisfacenti ai bisogni di elaborazione da loro direttamente individuati e definiti; ciò ha come conseguenza un'accresciuta, esplicita e consapevole dimensione socio-culturale dell'operazione di progettazione, realizzazione, diffusione dei singoli software e delle distribuzioni;

- la documentazione che accompagna i programmi, spesso inesistente, ha anch'essa un'impostazione dinamica e collaborativa: spesso si appoggia infatti sulla nascita e sulla crescita di comunità di persone che discutono sull'efficacia e sui possibili miglioramenti di un singolo programma o di un'intera distribuzione.

Inoltre bisogna considerare l'introduzione di nuovi modelli di business emergenti atti a "scoprire il codice" in cambio di agevoli fiscali. Tali vantaggi però si ripercuotono spesso sulla qualità del software, molte aziende che decidono di fornire il codice delle proprie applicazioni effettuano pesanti refactoring al fine di complicare notevolmente la comprensione del loro prodotto, costringendo praticamente chiunque volesse modificarlo ad appoggiarsi direttamente del servizio di assistenza fornito.

Perché un nuovo metodo?

Le ragioni complessive che hanno portato alla creazione di un nuovo metodo erano chiare fin dall'inizio: la valutazione effettuata attraverso la sola esperienza personale è sempre abbastanza imprecisa.

I metodi di valutazione Open Source sono ancora immaturi, ognuno di essi è focalizzato su diversi aspetti. In alcuni metodi esistenti, si valuta un elenco di indicatori prima di deciderne l'importanza, sprecando inutilmente tempo nello stimare elementi che poi non verranno presi in considerazione.

Nessun metodo di valutazione per prodotti Open Source tratta efficacemente le qualità interne ed esterne di prodotto pur avendo a disposizione il codice sorgente. Inoltre, non viene tenuta in considerazione la disponibilità di supporto nel tempo e il costo necessario per i moduli proprietari sviluppati da terze parti.

In seno alle considerazioni effettuate, si è giunti alla formulazione di un modello come unione ed estensione dell'Open BRR e del QSOS, introducendo nuove aree di valutazione e modificando la procedura in modo da considerare prima quali siano gli elementi da valutare assegnandogli un peso, quindi in base all'importanza attribuita, si valuterà quali saranno gli elementi da stimare.

Obiettivi

Gli utilizzatori di prodotti Open Source richiedono un metodo che consenta di selezionare in modo semplice quale software utilizzare.

A tal scopo, si vuole formulare un modello di valutazione semplice, oggettivo, valido e robusto in cui sia possibile sottoporre una lista di prodotti candidati in modo da ottenerne una griglia di valutazione finale.

Si vuole proporre l'utilizzo di un metodo aperto e standard che rispetti i principali requisiti di completezza, semplicità e adattabilità.

Completo: il requisito primario di ogni modello di valutazione è l'abilità dello stesso nell'evidenziare ogni caratteristica di prodotto, quale favorevole o no. E' quindi necessario che il punteggio assegnato ad ogni prodotto non sia mai fuorviante.

Semplice: per guadagnare una larga accettazione del metodo, il modello deve essere semplice da comprendere e da utilizzare.

Adattabile: dati i rapidi cambiamenti nell'industria del software, ogni modello di valutazione creato oggi potrebbe essere inutile in futuro. A tal scopo si vuole creare un modello aperto a possibili nuove estensioni.

Scopo principale del metodo e quello di approfondire alcuni aspetti fondamentali di un prodotto Open Source ossia:

- Adeguatezza funzionale: si vuole valutare se il prodotto in questione risponde ai requisiti funzionali richiesti.
- Qualità: assenza di bugs, tempo di risoluzione degli stessi
- Disponibilità di supporto nel tempo per attività di manutenzione correttiva, adattativa e percettiva.
- Costi: spese legate a moduli non O.S. o a tool di sviluppo necessari.
- Altri interessi: tipo di licenza, linguaggio di programmazione…

L’Open BQR è rivolto ad un target di lettori abbastanza ampio:

- Persone interessate ad approfondire nuove metriche per prodotto Open Source
- Comunità di progetti Open Source
- Esperti IT che vogliono ottenere, attraverso l’utilizzo del metodo, una valutazione ed una selezione tra prodotti Open Source omogenei.

Modello del processo di misurazione

Il processo di valutazione si compone di tre fasi:

1. Quick Assessment Filter
2. Data Collection & Processing
3. Data Translation

Fase 1: Quick Assessment Filter:

Come nel BRR, si identifica una lista di componenti da misurare ma, diversamente, le componenti individuate verranno valutate nella fase successiva, solo in seguito all’assegnazione di un fattore peso.

Si individuano cinque aree di indicatori da considerare:

1. Selezione di un insieme di indicatori basati su un target di utilizzo
2. Analisi delle qualità interne
3. Analisi delle qualità esterne
4. Disponibilità di supporto nel tempo
5. Verifica dei requisiti funzionali

Fase 1.1: Selezione di indicatori basati sull'ambito e sulle modalità di utilizzo

Obiettivo di tale fase è la valutazione di alcuni fattori di carattere generale, utili alla caratterizzazione finale dell'applicazione.

Si identifica inizialmente il target di utilizzo dell'applicazione (mission-critical, Regular, Development, Experimentation) quindi si passa alla valutazione del tipo di licenza: tale verifica è utile al fine di verificare se la licenza permette di svolgere l'attività richiesta dalle specifiche.

Successivamente si verificano altri parametri.

- Rispetto degli standard: se il rispetto degli standard industriali è cruciale per il tipo di software da valutare, si consiglia di tenere in considerazione tale parametro
- Linguaggio di implementazione: la pratica di adozione del software Open Source spesso richiede alcune attività di customizzazione e di supporto interno. Questo può essere importante nello scegliere applicazioni implementate in un linguaggio noto in azienda. Nel caso in cui si decida di assegnare tutto il processo di manutenzione alla software house che lo produce tale parametro può essere ininfluente.
- Supporto per l'internazionalizzazione o per la lingua desiderata: la mancanza o la presenza di tale supporto potrebbe essere fondamentale.
- Presenza di libri pubblicati sul prodotto: la disponibilità di libri sul prodotto selezionato è un indice importante del livello di maturità e del suo utilizzo quale indice di alta diffusione del prodotto oltre che di ampio interesse da parte della comunità.
- Il prodotto è seguito da analisti, tipo Gartner o IDC: un'altra indicazione importante è la disponibilità di reports del prodotto fatti da analisti professionisti.

Fase 1.2: Analisi delle qualità esterne

Si vuole verificare l'affidabilità delle applicazioni attraverso il livello di "difettosità". A tal scopo si analizza il database dei bugs in modo da determinare il rapporto di bugs risolti – scoperti ed il tempo medio di risoluzione dei bugs.

Infine, si verifica il numero di donazioni effettuate; si è riscontrato che la risoluzione di bugs porta spesso all'effettuazione di "donazioni" atte a diminuire il tempo di risoluzione.

E' importante sapere quanto rapidamente si sarà in grado di risolvere eventuali problemi. Una volta stabilito il limite di tempo, è possibile verificare se i tempi medi di risoluzione dei bug nel progetto considerato siano inferiori a quelli richiesti e verificare la percentuale di donazioni effettuate, necessarie a risolvere i bugs nel periodo stabilito.

Se il tempo medio di risoluzione è basso e le donazioni sono quasi inesistenti, il software è presumibilmente sviluppato da una comunità attiva di sviluppatori, altrimenti, in caso di tempi medi bassi ed elevato numero di donazioni, bisogna essere pronti ad effettuare donazioni di entità non determinata.

Fase 1.3: Analisi delle qualità interne

Il settore Industriale richiede prodotti maturi, completi e di qualità.

Dalla letteratura sulla "Software Metrics" si evince che attraverso un'analisi del software è possibile scoprire attraverso una serie di indicatori se un prodotto è ben progettato, manutenibile ed estensibile. Nello specifico, la Norma ISO 9126 ("Information Tecnology - Software product evaluation - Quality characteristics and guidelines for their use"), pubblicata nella sua prima versione nel 1991, definisce il modello dei requisiti qualitativi del Prodotto Software.

Da tale insieme, sono state selezionate solo le caratteristiche utili per una stima accurata e semplicemente misurabili.

Si distingue quindi un sottoinsieme delle qualità interne dello standard, composto dai parametri utili per la valutazione da parte delle aziende,e semplici da misurare.

Si richiede quindi di misurare la complessità ciclomatica (o numero ciclomatico di McCabe), utile a definire il livello di ramificazione dell'applicazione in modo automatizzato, il riuso effettuato e le dipendenze necessarie (database, moduli esterni…) utile a comprendere i possibili legami futuri con prodotti potenzialmente vulnerabili e il livello di modularità. Per definizione, un prodotto modulare è estensibile attraverso nuovi moduli, si ritiene tale caratteristica notevole nel caso di sviluppo interno del prodotto.

Fase 1.4: Disponibilità di supporto nel tempo

L'utilizzo di un prodotto, comporta la successiva manutenzione. La disponibilità di supporto nel tempo è un fattore molto importante.

Per questo, si sono introdotti alcuni nuovi indicatori: la verifica del numero di programmatori che rispondono alle richieste e delle aziende da cui provengono è indice di quanto il software è supportato. Più grande è la comunità, più facile sarà avere una risposta ai problemi futuri.

La verifica dell'attività di sviluppo, del numero di release rilasciate all'anno e della presenza di moduli aggiuntivi sviluppati da terze parti è una buona indicazione di un progetto attivo, di un prodotto in evoluzione.

Fase 1.5: Verifica delle funzionalità presenti

Da un analisi iniziale dei requisiti, è possibile estrarre tutte le funzionalità richieste rispetto ad un "uso standard". Una volta disponibile l'insieme di caratteristiche standard, sarà possibile valutarne l'importanza assegnando a ciascuna di esse un peso (da 0 a 9).

Infine bisogna creare una tabella dove inserire per ogni prodotto la percentuale di implementazione della funzionalità richiesta: zero per le funzionalità non implementate, 100 per funzionalità complete.

Fase 2: Data Collection & Processing

Terminata la fase precedente, si otterrà una tabella con tutti gli indicatori necessari ed un peso relativo assegnato. Conseguentemente sarà possibile eliminare tutti i campi con peso zero e, a propria discrezione, i campi con valori prossimi allo zero.

Per semplificare il processo di selezione, si è sviluppato uno strumento che permette l'inserimento e la visualizzazione dei pesi sotto forma di istogramma.

Una volta effettuata la valutazione dei pesi bisogna misurare tutte le caratteristiche di ogni singolo prodotto. La valutazione dell'importanza effettuata a priori, avrà sicuramente ridotto in modo notevole i tempi di stima mediante l'eliminazione delle caratteristiche non rilevanti.

Quindi, si normalizzano i pesi, ponendo la loro somma a cento. Infine si moltiplica il peso per il valore di importanza assegnato ottenendo il punteggio finale di ogni area.

E' quindi possibile ottenere un punteggio finale per ogni prodotto quale somma dei punteggi ottenuti per ogni area. Il punteggio così ottenuto sarà utile solo al fine di ottenere un rapido sguardo d'insieme, la valutazione complessiva andrà effettuata tramite le griglie ottenute in precedenza, coadiuvate dall'esperienza personale.

AREA DI VALUTAZIONE	PESO	STIMA	SCORE
Ambiti e modalità di utilizzo			
qualità esterne			
qualità interne			
supporto			
Funzionalità	%		

ESEMPIO DI GRIGLIA DI VALUTAZIONE

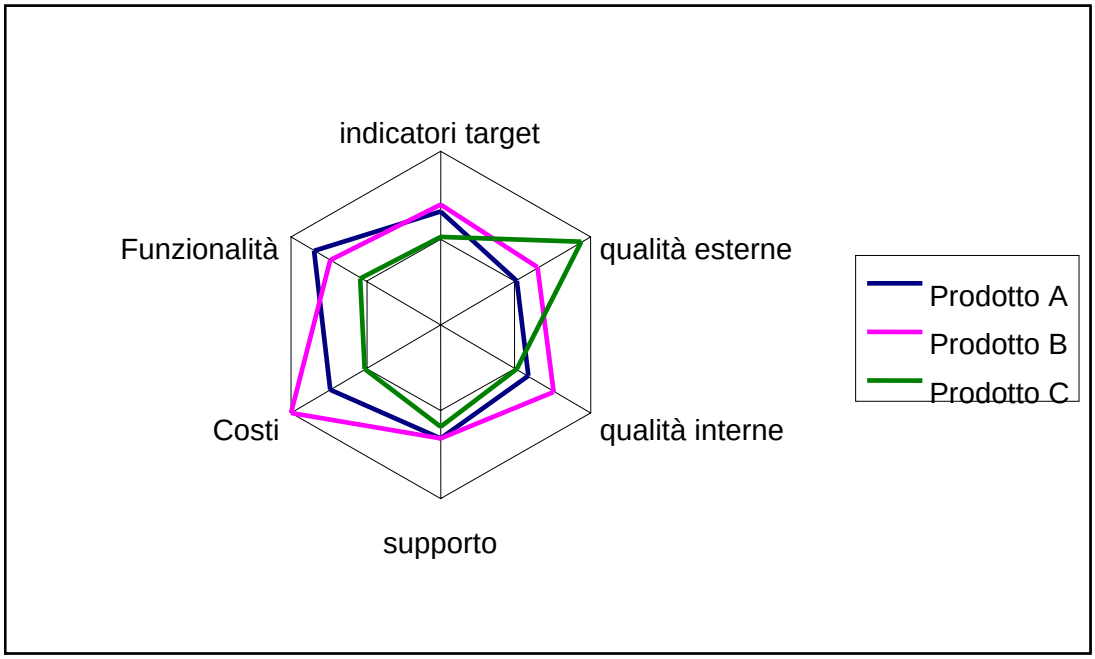
Fase 3: Data visualization

L'ultima fase consiste nella visualizzazione finale della griglia del punteggio delle caratteristiche per ogni prodotto.

Nel caso di una comparazione tra pochi prodotti, sarà possibile visualizzare il punteggio finale su un grafico di tipo radar.

AREA DI VALUTAZIONE	Prodotto A	Prodotto B	Prodotto C
indicatori target	20	25	10
qualità esterne	10	20	60
qualità interne	15	33	10
supporto	20	20	15
funzionalità	50	30	12
Open BQR	65	98	95

ESEMPIO DI VALUTAZIONE TRA TRE PRODOTTI



Open BQR schema riassuntivo

Indicatori	Peso assegnato	Valutazione indicatore
Target di utilizzo		
tipo di licenza		
rispetto degli standards		
linguaggio di implementazione		
supporto per internazionalizzazione		
libri sul prodotto		
seguito da analisti		
Analisi delle qualità esterne		
analisi database bugs		
rapporto bugs risolti/totale		
tempo medio risoluzione bugs		
rapporto donazioni/numero di bugs		
Analisi delle qualità interne		
complessità (Mc Cabe)		
riuso		
dipendenze		
Disponibilità supporto nel tempo		
analisi attività della comunità		
numero di release rilasciate		
numero di aziende che rispondono a richieste		
rapporto programmatori/azienda		
numero di programmatori indipendenti		
Verifica delle funzionalità presenti		

Open BQR Tool

A supporto degli utilizzatori dell'OpenBQR, è stato realizzato uno strumento di calcolo assistito attraverso due diverse forme: un foglio di calcolo MS Excel e un applicazione WEB.

Il sito web ufficiale, ospiterà tutta la documentazione relativa, esempi pratici, il whitepaper ed un forum atto ad ascoltare le necessità ed i problemi di tutti gli interessati.

Lo strumento è attualmente in fase di sviluppo, in tabella seguente, viene mostrato l'interfaccia MS Excel dello strumento.

			Open BQR	
Indicatori	**Peso assegnato**	**Valutazione indicatore**	**Peso Normalizzato**	**Valutazione complessiva**
Target di utilizzo				
tipo di licenza			0,00	0,00
rispetto degli standards			0,00	0,00
linguaggio di implementazione			0,00	0,00
supporto per internazionalizzazione			0,00	0,00
libri sul prodotto			0,00	0,00
seguito da analisti			0,00	0,00
Analisi delle qualità esterne (analisi database bugs)				
rapporto bugs risolti/totale			0,00	0,00
tempo medio risoluzione bugs			0,00	0,00
rapporto donazioni/numero di bugs			0,00	0,00
Analisi delle qualità interne				
complessità (Mc Cabe)			0,00	0,00
riuso			0,00	0,00
dipendenze			0,00	0,00
Disponibilità supporto nel tempo (analisi attività della comunità)				
numero di release rilasciate			0,00	0,00
numero di aziende che rispondono a richieste			0,00	0,00
rapporto programmatori/azienda			0,00	0,00
numero di programmatori indipendenti			0,00	0,00
		Totale	0,00	0,00

Schema riassuntivo	**Valutazione %**
Target di utilizzo	0,00
Analisi delle qualità esterne	0,00
Analisi delle qualità interne	0
Disponibilità supporto nel tempo	0
Verifica delle funzionalità presenti	90

OpenBQR.xls

Home Page del progetto

Comparazione

Criteria	OSMM Cap Gemini	OSMM® Navica	QSOS	OpenBRR	OpenBQR
Seniority	2003	2004	2004	2005	2006
Autori e sponsor	Cap Gemini	Navicasoft	Atos Origin	Carnegie Mellon West, SpikeSource, O'Reilly, Intel	Davide Taibi – Università dell'Insubria
Licenza	Non-free license, ma distribuzione autorizzata	Academic Free License	GNU	Creative Commons	Creative Commons
Assessment model	Practico	Practico	Practico	Scientifico	Scientifico
Livello di dettaglio	3 livelli	3 livelli	3 o più livelli	Due livelli	5 livelli
Criteri tecnici/funzionali	No	No	Si	Si	Si
Scoring model	Flessibile	Flessibile	Flessibile	Rigido	Flessibile
Scoring scale by criterion	1 - 5	1 -10	0 -2	1 -5	1-100
Iterative process	No	No	Si	Si	Si
Criteria weighting	Si	Si	Si	Si	Si

OSMM - NAVICA	QSOS	Open BRR	Open BQR
Phase 1: Assessment Element Maturity	**Phase 1: Definition**	**Phase 1: quick assessment**	**Phase 1: quick assessment**
		Target di utilizzo	Ambito di utilizzo
Product software	tipo di licenza	tipo di licenza	tipo di licenza
support	software families	rispetto degli standards	rispetto degli standards
documentation	tipo di comunità	esistenza utenti referenziabili	linguaggio di implementazione
training		supporto fornito da organ. Stabile	supporto per internazionalizzazione
product integration	**Phase 2: Evaluation**	linguaggio di implementazione	libri sul prodotto
professional services	Informazioni generali (descrizione, riferimenti…)	supporto per internzionalizzazione	seguito da analisti
	Servizi esistenti	presenza moduli aggiuntivi di terze parti	
	documentazione	libri sul prodotto	Analisi delle qualità esterne
	offerte di supporto commerciale	seguito da analisti	analisi database bugs
	numero di offerte di training		rapporto bugs risolti/totale
	numero di consulenti		tempo medio risoluzione bugs
	modularity		rapporto donazioni/numero di bugs
			Disponibilità supporto nel tempo
			analisi attività della comunità
			numero di release rilasciate
			numero di aziende che rispondono a richieste
			rapporto programmatori/azienda
			numero di programmatori indipendenti
			Verifica del codice
			complessità (Mc Cabe)
			riuso
			dipendenze
			Analisi dei costi
			Verifica delle funzionalità presenti
Phase 2: Assign weighting factor	**Phase 3: Qualification**	**Phase 2: target usage assessment**	**Phase 2: Data collection and processing**
Assegnamento pesi	Assegnamento pesi	Assegnamento pesi	Assegnamento pesi
	Definizione di vincoli di accettazione	ordinamento prodotti	Normalizzazione punteggio
			Valutazione componenti
Phase 3: Calcolate maturity score	**Phase 4: Selection**	**Phase 3: data collection and processing**	**Phase 3: data visualization**
	Strict selection	Comparazione caratteristiche	visualizzazione griglia punteggio
	Loose selection	Assegnamento pesi	grafico radar
		ordinamento prodotti	

Capitolo

6

Open BQR case studies

L'Open BQR, per sua natura, è stato concepito come metodo per la comparazione di più prodotti. Il metodo è applicabile anche per la valutazione di un solo prodotto di qualunque dimensione e complessità.

A dimostrazione delle capacità dell'Open BQR, si introducono due diverse valutazioni: la prima, relativa alla creazione di tre prodotti CMS relativamente semplici, per la realizzazione di un semplice sito web personale, la seconda per la valutazione di un singolo prodotto ERP di dimensioni considerevoli.

Caso 1: Content Management System comparison

Si vuole realizzare un'applicazione web personale, con layout definito dal cliente.

Dovrà essere possibile creare nuove pagine pubbliche e nascoste da parte dell'utente finale. Si richiede una galleria immagini ed una pagina contenente più file da poter scaricare. I file disponibili dovranno essere caricati su web attraverso il sistema e con la possibilità di rimuoverli in un secondo tempo.

Si richiede inoltre che il pannello di amministrazione sia possibilmente in Italiano

Dalla descrizione sintetica riportata è possibile visualizzare le specifiche sotto forma di elenco da assegnare un punteggio relativo all'importanza che ricopre nell'applicazione:

1. Possibilità di creazione layout personalizzato 10
2. CRUD (Create, Read, Update, Delete) delle pagine da parte dell'utente 10
3. Galleria immagini 7

4. CRUD (Create, Read, Update, Delete) file e pagina di download 5
5. Supporto Lingua Italiana 4

A tal scopo si vuole vagliare la possibilità di utilizzare un prodotto CMS (Content Management System) Open Source, al posto della classica realizzazione ad hoc.

CMS (Content Management System)

Con la veloce e capillare diffusione di internet sono stati realizzati un gran numero di siti che troppo somigliano ad un volantino di carta.
Il risultato è che il web è pieno di siti "relitto" che non danno alcun valore aggiunto e che a volte rischiano addirittura di dare un'immagine distorta dell'azienda che dovrebbero rappresentare.

Il reale valore aggiunto di un sito internet dovrebbe essere il costante aggiornamento dei contenuti. Solo un sito costantemente aggiornato può essere seguito con interesse. Facciamo qualche esempio: un attività commerciale che decide di proporre un'offerta promozionale ha la necessità di pubblicizzare l' evento sul proprio sito, ma come fare ad avere dei tempestivi aggiornamenti? Oppure, un’associazione vuole aggiornare l'elenco degli iscritti o inserire delle news nel proprio sito, o ancora, un'agenzia di viaggi vuole pubblicizzare le proprie offerte.

Normalmente, per queste operazioni, come per ogni seppur minimo aggiornamento, si seguono due diverse procedure: nel caso il sito sia realizzato in modo statico, l'interessato (autore, o content manager) fornisce i contenuti da pubblicare a chi ha realizzato le pagine (web master) e dopo aver spiegato nel migliore dei modi ciò che desidera, aspetta diligentemente il suo turno. Ovviamente non sarà il solo utente che ha la necessità di aggiornare il proprio sito, quindi l'operazione sarà lenta, dispendiosa (gli aggiornamenti si pagano!) e soprattutto genera un rapporto di dipendenza nei confronti del web master o della web agency che ha realizzato il sito. Nel caso invece di un sito dinamico, il cliente potrà personalizzare tutti i contenuti che sono stati resi modificabili,

ma, conseguentemente essendo il sito realizzato ad hoc, i costi iniziali saranno molto più alti.

Il CMS, invece è un'ottima soluzione alternativa, economicamente competitiva, che consente di essere autonomi e tempestivi negli aggiornamenti e nella creazione di nuove pagine.

Il CMS è un sistema di gestione dei contenuti che permette ad un content manager o all'autore, che potrebbe non conoscere l'HTML, di gestire la creazione, la modifica e la rimozione di contenuti da un sito web senza aver bisogno di conoscenze di programmazione specifiche. In poche parole, garantisce una totale autonomia nella gestione di un sito web.

La scelta dei prodotti

Il panorama dei CMS è molto dinamico: il numero di prodotti che si spartiscono il mercato è elevatissimo. Ci sono pacchetti però, che hanno una lunga tradizione alle spalle, sulla scorta anche di fork di progetti; altri prodotti sono approdati sul mercato solo da qualche anno ma sono ugualmente validi e diffusi.

Il criterio di scelta è stato molto semplice: sono stati inclusi nella prova i tre progetti CMS più attivi adatti alla creazione di siti web adatti alla creazione di siti web personali.

I prodotti che sono quindi rientrati nella comparativa portano nomi probabilmente noti a tutti i professionisti IT: si tratta di Mambo (http://www.mamboserver.com), Drupal (http://www.drupal.org) e WebGUI (http://www.webgui.it).

Mambo

Mambo è un content management system, che permette di scrivere, organizzare, modificare, pubblicare informazioni e notizie attraverso un editor WYSIWYG integrato nel pannello di amministrazione.

Il cms è formato dal core (il nucleo), e da un insieme di elementi aggiuntivi, denominati componenti, moduli, mambots e templates, che ne permettono l'espansione.

I componenti permettono di aggiungere nuove funzionalità al cms: gallerie di immagini, forum, download, statistiche, ecc.

I moduli consentono di creare dei menù, blocchi di testo, immagini ed altro. Di solito richiamano dal database dei dati riguardanti gli elementi installati, ma possono essere costituiti anche da elementi statici.

Data la - relativa - facilità di realizzazione di queste espansioni, che molto spesso sono messe a disposizione della comunità internazionale a titolo gratuito, il cms si contraddistingue per l'adattabilità ai contesti più diversi.

La comunità di sviluppo pone particolare attenzione alla sicurezza. Non appena si scorge una potenziale falla, il team di sviluppo si attiva tempestivamente per porvi rimedio.

Mambo ha inoltre un'articolata interfaccia di amministrazione che, una volta apprese le nozioni fondamentali sulla struttura del CMS, risulta essere di facile navigazione anche per i principianti.

L'installazione è molto semplice in quanto effettuata tramite interfaccia web.

Drupal

Drupal è un CMS, ovvero un gestore di contenuti e di siti Web dinamici realizzato in PHP. Con Drupal è possibile realizzare diversi tipi di siti Web o intranet, per pubblicare articoli, insiemi di messaggi/commenti, forum di discussione, blog, raccolte di immagini etc.

Drupal consente agli utenti di registrarsi e autenticarsi in modo da tenere traccia di chi è autore di ogni singolo contenuto, e permettere agli amministratori di consentire livelli di accesso differenziati a seconda dei ruoli (utente, moderatore, amministratore, etc.), consente di organizzare i contenuti in base alla tipologia (pagina, messaggio del forum, immagine, etc) e alla categoria assegnata dall'amministratore: una singola pagina può essere per esempio classificata come articolo, documentazione, descrizione prodotto, etc. Questo consente di dividere i contenuti in modo estremamente flessibile, rendendone semplice l'inserimento e la visualizzazione, e consentendo di realizzare uno schema di navigazione del sito estremamente funzionale.

Punti di forza di Drupal sono sicuramente l'ampia flessibilità e configurabilità, la robustezza e la gestione della sicurezza. Drupal è realizzato in modo modulare, consentendo di aggiungere numerose funzionalità aggiuntive al sistema di base.

Drupal è un applicazione Web. Questo significa che i contenuti vengono inseriti e visualizzati attraverso un Web browser (Internet Explorer, Mozilla Firefox, Opera etc.). E' possibile realizzare siti Web pubblici o locali (intranet), a patto di disporre di un server in grado di pubblicare pagine Web con interprete PHP e di un database SQL Il database predefinito è MySQL, ma praticamente qualsiasi database supportato da PHP è utilizzabile.

La comunità di sviluppo è abbastanza attiva, anche se non comparabile a quella di Mambo.

L'installazione, è molto semplice ma non completamente automatizzata come quella do Mambo, in quanto necessita di alcune modifiche ai files di configurazione.

Web Gui

WebGUI è una piattaforma di content management che consente di creare e aggiornare siti web anche complessi. E' modulare, scalabile, personalizzabile ed installabile su diverse piattaforme e soprattutto semplice da usare.

Queste caratteristiche ne fanno lo strumento ideale anche per esigenze piú sofisticate quali possono essere ad esempio quelle di un portale oppure quelle di una Intranet o Extranet aziendale per arrivare ad applicazioni di Cooperative working e di E-learning .

WebGUI è un application framework (struttura di applicazioni) che gestisce il content management.

Punto forza dell'applicativo è la modularità: una volta che la tua applicazione è realizzata, può essere riutilizzata tutte le volte che serve in differenti sezioni del sito. Per esempio, una volta creata un'applicazione per un forum, è possibile inserirla in varie sezioni differenti del sito, cosa non possibile con Mambo e Drupal dove si è costretti a reindirizzare gli utenti alla "sezione forum" del sito.

Grosso svantaggio è quello della difficoltà di installazione: è quasi impossibile installarlo su molti server, in quanto richiede una personalizzazione dei permessi, installazione di moduli perl ed altre modifiche manuali non consentite dalla maggior parte di provider, a patto di non acquistare un server dedicato.

La comunità di sviluppo è abbastanza sviluppata anche se non paragonabile a quella di Mambo e Drupal. Il supporto commerciale è molto economico ed è fornito da piccole società dislocate in tutto il mondo, di cui una anche in Italia.

Open BQR – Applicazione

Prima di analizzare tutti i prodotti, si definisce l'insieme dei pesi da utilizzare.

Indicatori	Peso assegnato
Target di utilizzo	
tipo di licenza	9
rispetto degli standards	0
linguaggio di implementazione	0
supporto per internazionalizzazione	4
libri sul prodotto	2
seguito da analisti	0
Analisi delle qualità esterne (analisi database bugs)	
rapporto bugs risolti/totale	6
tempo medio risoluzione bugs	4
rapporto donazioni/numero di bugs	6
Analisi delle qualità interne	
complessità (Mc Cabe)	0
riuso	0
dipendenze	0
Disponibilità supporto nel tempo (analisi attività della comunità)	
numero di release rilasciate	9
numero di aziende che rispondono a richieste	5
rapporto programmatori/azienda	4
numero di programmatori indipendenti	4

Considerato il tipo di applicazione da realizzare, risulta fondamentale il tipo di licenza, dovrà essere possibile utilizzare un prodotto CMS senza però doverne indicare i riferimenti. Il supporto per l'internazionalizzazione, è un requisito non fondamentale, mentre la presenza di libri sul prodotto risulta essere di minima importanza quale supporto allo sviluppo.

Si ritiene fondamentale l'assenza di bugs ed una buona assistenza da parte della comunità.

Non vengono considerate le qualità del codice, non essendo necessario accedere ai sorgenti. Punto fondamentale di un prodotto CMS è proprio la possibilità di personalizzazione modifiche di codice.

In seguito vengono elencate le caratteristiche tecniche riassuntive di ciascun prodotto preso in considerazione.

Product	**Drupal 4.7.4**	**Mambo 4.5.3**	**WebGUI 7.0**
System requirements	Drupal	Mambo	WebGUI
Application Server	PHP 4.3.3+	PHP 4.1.2+	mod_perl
Costo	Free	Free	Free
Database	MySQL, Postgres	MySQL	MySQL
License	GNU GPL	GNU GPL	GNU GPL
Operating System	Any	Any	Any
Programming Language	PHP	PHP	Perl
Web Server	Apache, IIS	Apache, IIS, any PHP enabled web server	Apache
Support	Drupal	Mambo	WebGUI
Commercial Manuals	Yes	Yes	Yes
Commercial Support	Yes	Yes	Yes
Commercial Training	Yes	Yes	Yes
Developer Community	Yes	Yes	Yes
Online Help	Yes	Yes	Yes
Public Forum	Yes	Yes	Yes
Third-Party Developers	Yes	Yes	Yes
Ease of Use	Drupal	Mambo	WebGUI
Mass Upload	Free Add On	No	Yes
Prototyping	No	No	Yes
Server Page Language	Yes	Yes	Yes
Spell Checker	Free Add On	No	Limited
Style Wizard	No	No	Yes
Template Language	Limited	Yes	Yes
WYSIWYG Editor	Free Add On	Yes	Yes
Built-in Applications	Drupal	Mambo	WebGUI
Blog	Yes	Yes	Yes
Document Management	Limited	Free Add On	Limited
File Distribution	Free Add On	Free Add On	Yes
Link Management	Free Add On	Yes	Yes
Mail Form	Free Add On	Yes	Yes
Photo Gallery	Free Add On	Free Add On	Yes

OpenBQR - Tabelle di Valutazione

Mambo

Indicatori	Peso assegnato	Valutazione indicatore	Valutazione comple
Target di utilizzo			
tipo di licenza	9	10	15,52
rispetto degli standards	5	8	6,90
linguaggio di implementazione	0	0	0,00
supporto per internazionalizzazione	4	10	6,90
libri sul prodotto	2	4	1,38
seguito da analisti	0	0	0,00
Analisi delle qualità esterne (analisi database bugs)			
rapporto bugs risolti/totale	6	8	8,28
tempo medio risoluzione bugs	4	5	3,45
rapporto donazioni/numero di bugs	6	0	0,00
Analisi delle qualità interne			
complessità (Mc Cabe)	0	0	0,00
riuso	0	0	0,00
dipendenze	0	0	0,00
Disponibilità supporto nel tempo (analisi attività della comunità)			
numero di release rilasciate	9	10	15,52
numero di aziende che rispondono a richieste	5	9	7,76
rapporto programmatori/azienda	4	9	6,21
numero di programmatori indipendenti	4	9	6,21
		Totale	78,10

Schema riassuntivo	Valutazione %
Target di utilizzo	30,69
Analisi delle qualità esterne	11,72
Analisi delle qualità interne	0
Disponibilità supporto nel tempo	35,6896552
Verifica delle funzionalità presenti	100

Drupal

Indicatori	Peso assegnato	Valutazione indicatore	Valutazione complessiva
Target di utilizzo			
tipo di licenza	9	10	15,52
rispetto degli standards	5	8	6,90
linguaggio di implementazione	0	0	0,00
supporto per internazionalizzazione	4	10	6,90
libri sul prodotto	2	0	0,00
seguito da analisti	0	0	0,00
Analisi delle qualità esterne (analisi database bugs)			
rapporto bugs risolti/totale	6	6	6,21
tempo medio risoluzione bugs	4	5	3,45
rapporto donazioni/numero di bugs	6	0	0,00
Analisi delle qualità interne			
complessità (Mc Cabe)	0	0	0,00
riuso	0	0	0,00
dipendenze	0	0	0,00
Disponibilità supporto nel tempo (analisi attività della comunità)			
numero di release rilasciate	9	8	12,41
numero di aziende che rispondono a richieste	5	5	4,31
rapporto programmatori/azienda	4	9	6,21
numero di programmatori indipendenti	4	9	6,21
		Totale	68,10

Schema riassuntivo	Valutazione %
Target di utilizzo	29,31
Analisi delle qualità esterne	9,66
Analisi delle qualità interne	0
Disponibilità supporto nel tempo	29,14
Verifica delle funzionalità presenti	100

Web GUI

Indicatori	Peso assegnato	Peso Normalizzato	Valutazione complessiva
Target di utilizzo			
tipo di licenza	9	15,52	15,52
rispetto degli standards	5	8,62	6,03
linguaggio di implementazione	0	0,00	0,00
supporto per internazionalizzazione	4	6,90	6,90
libri sul prodotto	2	3,45	0,00
seguito da analisti	0	0,00	0,00
Analisi delle qualità esterne (analisi database bugs)			
rapporto bugs risolti/totale	6	10,34	9,31
tempo medio risoluzione bugs	4	6,90	4,83
rapporto donazioni/numero di bugs	6	10,34	0,00
Analisi delle qualità interne			
complessità (Mc Cabe)	0	0,00	0,00
riuso	0	0,00	0,00
dipendenze	0	0,00	0,00
Disponibilità supporto nel tempo (analisi attività della comunità)			
numero di release rilasciate	9	15,52	7,76
numero di aziende che rispondono a richieste	5	8,62	2,59
rapporto programmatori/azienda	4	6,90	1,38
numero di programmatori indipendenti	4	6,90	1,38
		100,00	55,69

Schema riassuntivo	Valutazione %
Target di utilizzo	28,45
Analisi delle qualità esterne	14,14
Analisi delle qualità interne	0
Disponibilità supporto nel tempo	13,10344828
Verifica delle funzionalità presenti	100

Comparazione finale

	Mambo	Drupal	Web GUI
Target di utilizzo	30,69	20,31	28,45
Analisi delle qualità esterne	11,72	9,66	14,14
Disponibilità supporto nel tempo	35,68	29,14	13,10
Verifica delle funzionalità presenti	100	100	100
Valutazione	78,10%	68,10	55,69
	☆☆☆☆	☆☆☆	☆☆

Considerazioni:

La comparazione effettuata, indica chiaramente Mambo quale soluzione più adatta alla realizzazione del sito indicato nella descrizione delle specifiche.

Tale considerazione è pienamente compatibile alle conclusioni tratte dalle comparazioni effettuate dalla comunità degli utilizzatori di prodotti CMS quali opensourcecms.com, cmsjournal.com e cmsmatrix.org, dove vengono classificati nello stesso ordine per la realizzazione di siti web molto semplici.

La comparazione verrà estesa nei prossimi sei mesi ad un test molto più accurato a cui prenderanno parte numerosi gruppi di sviluppo assegnando a tutti i gruppi la definizione delle specifiche di un sito web mediamente complesso. Alcuni gruppi effettueranno la valutazione OpenBQR, mentre i team rimanenti realizzeranno l'applicazione web con un gruppo di CMS indicati, allo scopo di verificare, date le specifiche, il tempo di customizzazione, le difficoltà incontrate, e la qualità finale di realizzazione.

Tale studio sarà utile a capire quali sono i parametri più utili ed a raffinare il livello di specifica dell'insieme dei pesi.

Caso 2: ERP

L'Open BQR è stato studiato quale sistema di selezione per più prodotti, tale modello risulta utilizzabile anche per la valutazione di singoli prodotti di qualsiasi dimensione.

A tal scopo, si effettua la stima di un prodotto per definizione molto complesso, un ERP Open Source: "Compiere".

Si vuole realizzare, per conto terzi, un prodotto in grado di gestire i seguenti compiti:

- amministrativi e finanziari, gestione delle vendite e degli acquisti
- supporto per la gestione della relazione con i clienti (CRM) e risorse umane organizzazione

- gestione e ottimizzazione della logistica e distribuzione pianificazione e controllo della produzione

Si necessita che l'applicativo sia tradotto in lingua Italiana e, per motivi di competenze degli sviluppatori, che sia scritto in Java

Compiere: ERP Open Source

Si tratta di un applicativo server, sviluppato totalmente in Java dalla ComPiere Inc.

Si avvale dell'ausilio di Oracle 9i come DBMS, utilizzando l'infrastruttura concessa da server applicativi ben più complessi ,come JBoss e Tomcat.

Analisi dell'applicativo

Utilizzando le definizioni normalmente usate in ambito commerciale per i sistemi informatici aziendali, Compiere, quale prodotto ERP, fornisce in maniera integrata funzionalità proprie in almeno cinque ambiti differenti:

- Customer Relations Management (CRM), gestione delle relazione con i clienti;
- Enterprise Resource Planning (ERP), gestione della risorse aziendali;
- Online Analysis Processing (OLAP), analisi in tempo reale dell'andamento dell'azienda;
- Partner Relations Management (PRM), gestione della relazione con i partner;
- Supply Chain Management (SCM), gestione della catena di approvvigionamento.

Queste funzionalità sono offerte in maniera completamente integrata in una serie di moduli:

- Quote-to-Invoice, dal preventivo alla fatturazione di vendita;
- Requisition-to-Invoice, dalla richiesta alla fatturazione di acquisto;

- Material Management (Supply Chain Management), la gestione dei prodotti e dei magazzini;
- Partner Relations Management, la gestione delle relazione con i business partner (clienti, fornitori, dipendenti, collaboratori, . . .);
- Performance Analysis, l'analisi delle prestazioni economiche (contabilità e gestione dei centri di costo);
- Project Management, la gestione dei progetti aziendali;
- Web Store, per la gestione del commercio elettronico via web.

Bisogna però aggiungere che questi moduli sono strettamente integrati fra di loro, e che Compiere non presenta tutte quelle problematiche di sincronizzazione comuni a molte di quelle soluzioni software che sono state sviluppate come un'integrazione a posteriore di strumenti informatici già esistenti. Compiere è infatti un sistema informatico dall'aspetto piuttosto monolitico ma dotato comunque di una notevole flessibilità.

I moduli non sono semplicemente distinguibili in quanto integrati tra loro, conseguentemente non è stato possibile valutarne la complessità e la qualità del codice separatamente. Sono state quindi calcolate alcune metriche su tutto il prodotto comprensivo dei sette moduli utilizzando "CodePro Studio"[15] ed eclipse[16]. Il report delle metriche calcolate è disponibile in appendice A.

Esistono numerose possibilità di supporto a livello locale, in Italia ad oggi, due aziende forniscono assistenza sotto forma di consulenze, corsi o sviluppo in outsourcing.

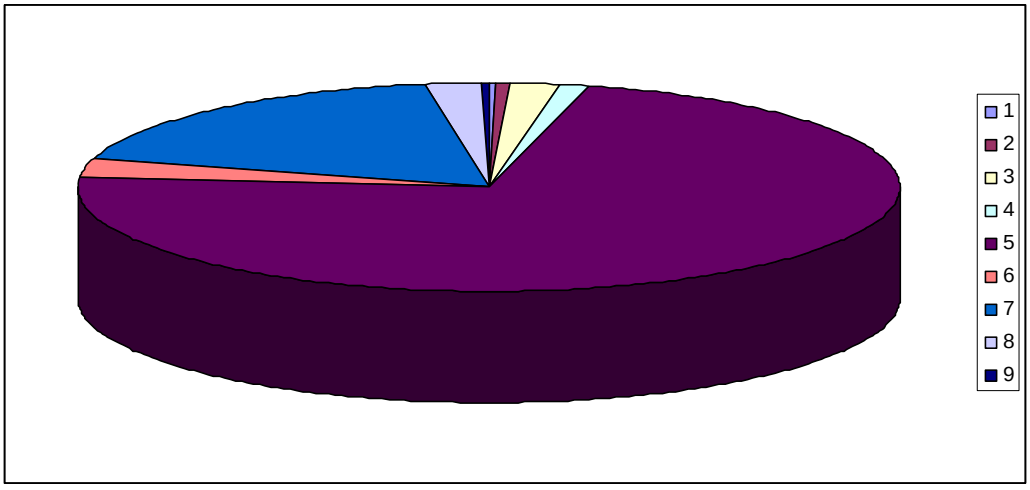
L'assistenza e la risoluzione dei bugs è effettuata solamente dalla ComPiere Inc in tempi abbastanza brevi, buona parte delle richieste è accompagnata da una donazione, suddivisa in cinque livelli di prezzo non specificati. Buona parte delle donazioni, sono di massimo livello.

I bugs sono divisi in una scala di priorità da un minimo di 1 ad un massimo di 9, sono stati rilevati il 72,2% dei bugs totali di scala 5.

E’ stata effettuata un’analisi approfondita del database dei bug di Compiere, dove sono stati risolti oltre l’85% dei bugs rilevati.

Al momento ci sono 330 bugs aperti su 2292 così suddivisi:

priorità	# BUG TOTALI	#BUG RISOLTI	Commenti (media per bug)	giorni risoluzione (media)	Percentuale bug / priorità
1	6	6	1	3	0,2
2	13	13	4,5	14	0,5
3	45	39	3,2	22	2
4	26	22	2,7	21,7	1,1
5	1656	1410	4,1	21,7	72,2
6	72	60	5,4	20,2	3,1
7	412	357	5,9	44,5	18
8	52	51	7,6	35,25	2,2
9	6	6	12	12	0,2
TOTALE	2292	1962	5	22	100



Numero di bugs per livello

L’azienda dà la possibilità di aprire richieste di supporto tecnico oltre a richieste di sviluppo di nuove funzionalità, che se d’interesse comune, possono essere implementate gratuitamente.

Al momento sono stati aperti 3816 post per richieste di supporto tecnico di cui 3793 hanno avuto risposta.

Sono state richieste dalla comunità 602 nuove caratteristiche di cui 287 sono state implementate. Per tutte le nuove caratteristiche richieste e successivamente realizzate è stata versata una "donazione".

Da un analisi successiva, si è notata una certa tendenza all'accettazione di donazioni che, in caso di bug, riducono notevolmente il tempo di attesa, contenendo i tempi di risoluzione in un massimo di sette giorni.

Open BQR – Applicazione

Indicatori	Peso assegnato	Valutazione indicatore	Valutazione comple
Target di utilizzo			
tipo di licenza	9	10	8,26
rispetto degli standards	6	8	4,40
linguaggio di implementazione	10	0	0,00
supporto per internazionalizzazione	10	10	9,17
libri sul prodotto	2	4	0,73
seguito da analisti	5	0	0,00
Analisi delle qualità esterne (analisi database bugs)			
rapporto bugs risolti/totale	8	8	5,87
tempo medio risoluzione bugs	9	5	4,13
rapporto donazioni/numero di bugs	10	0	0,00
Analisi delle qualità interne			
complessità (Mc Cabe)	6	0	0,00
riuso	6	0	0,00
dipendenze	6	0	0,00
Disponibilità supporto nel tempo (analisi attività della comunità)			
numero di release rilasciate	9	10	8,26
numero di aziende che rispondono a richieste	5	9	4,13
rapporto programmatori/azienda	4	9	3,30
numero di programmatori indipendenti	4	9	3,30
		Totale	51,56

Schema riassuntivo	Valutazione %
Target di utilizzo	22,57
Analisi delle qualità esterne	10,00
Analisi delle qualità interne	0
Disponibilità supporto nel tempo	18,9908257
Verifica delle funzionalità presenti	100

Capitolo

7

Conclusioni

This is not the end,

Even not the beginning of the end,

but it may be the end of the beginning.

Winston Churchill

Winston Churchill non pronunciò certo questa frase riferendosi allo stato di avanzamento delle metriche del Software Open Source; sta di fatto che, questo suo pensiero, può essere adottato per descrivere lo sforzo che il mondo sta facendo per poterlo integrare all'interno delle normali metodologie di stima del software. Da quando sono stati approntati i primi modelli di metriche, molti sono stati gli sforzi sostenuti dalla comunità Open Source.

Ciò nonostante, tutte queste innovazioni, non costituiscono altro che l'inizio o, meglio, la fase di predisposizione degli strumenti che potranno essere utilizzati per comprendere e valutare al meglio i prodotti Open Source.

Una volta che questa fase raggiungerà la propria fine, si aprirà finalmente la fase in cui, avendo a disposizione degli strumenti efficienti, sicuri ed univoci, molti adotteranno queste metodologie certi di poterne trarre dei vantaggi economici per la propria attività commerciale.

Il lavoro qui presente, che è appena giunto alla sua conclusione, è stato realizzato con lo scopo di descrivere lo stato dell'arte in cui si trova, ad oggi, l'universo delle metriche del software Classiche e dei modelli di comparazione per Prodotti Open Source.

Dopo lo studio delle metriche classiche, è stato formulato un nuovo modello di comparazione Open Source come estensione ed integrazione dei modelli esistenti.

Ancora molto deve essere fatto prima che veramente sia possibile dire di aver realizzato un modello, integrato ed universalmente riconosciuto.

Il lavoro svolto nel corso della tesi ha prodotto dei risultati conformi alle aspettative ed agli obiettivi prefissati.

Lo studio delle Metriche del Software, mi ha permesso di venire a contatto con un mondo ancora sconosciuto ed estremamente interessante.

Lavorando a stretto contatto con la comunità del Software Metrics e dell'Open Source, seguendone gli incontri e ponendo loro svariati quesiti, si è potuti venire a conoscenza delle loro esigenze per quanto riguarda l'utilizzo e l'apprezzamento delle metriche del software in ambito Open Source. Basandoci quindi sulle richieste, sulle incomprensioni e sui problemi stessi, sono stati definiti i requisiti da sviluppare, relativi a problemi riscontrati da implementare, ma anche alla correzione di possibili imperfezioni rilevate.

Maggior attenzione è stata posta sulla verifica dei problemi riscontrati nei metodi di valutazione già esistenti, ottenendo un nuovo modello migliorativo, basandosi principalmente sulla semplicità piuttosto che sull'aspetto della presentazione all'utente.

Dall'analisi effettuata su casi d'uso differenti, si potuto notare la semplicità di applicazione del modello ad un ampio spettro di prodotti.

Sviluppi futuri

Il modello è sicuramente perfettibile; nei prossimi mesi verrà applicato in larga scala ad un progetto CMS, allo scopo di validarne le funzionalità.

Il prossimo obiettivo è già stato fissato nello stabilire quali siano i parametri più significativi da valutare, quindi si passerà allo studio di un sistema per rendere la valutazione effettuata assoluta e priva di possibili interpretazioni personali.

Ulteriore sviluppo è la sensibilizzazione della comunità O.S. per poter accedere in maniera più veloce al database dei bugs dei progetti Open Source o, in via alternativa, avere a disposizione sul sito di ogni progetto delle statistiche relative al numero di bugs aperti e risolti, al numero di aziende e programmatori coinvolti e del numero di release rilasciate.

Bibliografia

1. Function Point: Manuale delle Regole di Conteggio Versione 4.2 (IFPUG)
2. Function Point Analysis – Measurement Practices for successful Software Project (Garmus, Herron)
3. Best Practices in Software Measurement (Ebert et all)
4. Software Metrics a guide to planning Analysis and application (Pandian)
5. Metrics and Models in Software Quality Engineering (H.Kan)
6. IT Measurement – Practical Advice from the Experts (IFPUG – Yourdon)
7. Valutazione di ERP basati su Function Points (A. Cavallo, M. Martellucci, F. M. Stilo, N. Lucchetti e D. Natale)
8. Software Complexity." Crosstalk, Journal of Defense Software Engineering (McCabe, Thomas J. & Watson, Arthur H)
9. Best Practices in Software Measurement: How to Use Metrics to Improve Project and Process Performance (C. Ebert, R. Dumke, M. Bundschuh)
10. Misurare il software. Quantità, qualità, standards e miglioramento di processo nell'Information Technology (Luigi Buglione)
11. Metriche del software Esperienze e ricerche (GUFPI-ISMA)

Sitografia

12. http://www.cwi.it/showPage.php?template=articoli&id=13550
13. http://www.dwheeler.com/oss_fs_why.html#tco
14. http://en.wikipedia.org/wiki/Total_cost_of_ownership
15. http://www.instantiations.com/codepro/index.html
16. http://www.eclipse.org
17. http://www.capgemini.com
18. http://www.navicasoft.com
19. http://www.atosorigin.com/
20. http://www.ifpug.org/
21. http://www.nesma.org
22. http://www.cosmicon.com
23. http://www.geronesoft.com
24. http://www.gufpi-isma.org

Appendice A

Metric Name	Value
Abstractness	3.4%
Average Block Depth	0.59
minimum	0
maximum	9
Average Cyclomatic Complexity	1.59
minimum	1
maximum	115
Average Number of Constructors Per Type	1.49
minimum	0
maximum	11
Average Number of Fields Per Type	3.32
minimum	0
maximum	85
Average Number of Methods Per Type	13.30
minimum	0
maximum	147
Average Number of Parameters	0.67
minimum	0
maximum	18
Comments Ratio	22.2%
Difficulty	75,055.61
Distance	0.1
Efferent Couplings	1,425
Effort	930,544,816,818.35
Instability	0.98
Lines of Code	221,528
Number of Characters	10,876,799
Number of Comments	49,300
end-of-line	21460
multi-line	175
javadoc	27665
Number of Constructors	2,152
public	2079
protected	23
package	25
private	25
Number of Fields	8,303
instance	4412

public	255
protected	128
package	121
private	3908
static	3891
public	3197
protected	48
package	134
private	512
public	3452
instance	255
static	3197
protected	176
instance	128
static	48
package	255
instance	121
static	134
private	4420
instance	3908
static	512
Number of Lines	384,479
Number of Methods	19,161
instance	18197
public	16357
protected	800
package	117
private	923
static	964
public	855
protected	20
package	12
private	77
public	17212
instance	16357
static	855
protected	820
instance	800
static	20
package	129
instance	117
static	12
private	1000

instance	923
static	77
Number of Operands	474,305
Number of Operators	199,743
Number of Packages	95
compilation units	1382
minimum	0
average	14
maximum	451
class files	0
minimum	0
average	0
maximum	0
Number of Semicolons	101,090
Number of Types	1,440
interface	31
public	31
protected	0
package	0
private	0
class	1409
public	1357
protected	0
package	46
private	6
public	1388
interface	31
class	1357
protected	0
interface	0
class	0
package	46
interface	0
class	46
private	6
interface	0
class	6
Number of Unique Operands	261,556
Number of Unique Operators	82,779
Program Length	674,048
Program Vocabulary	344,335
Program Volume	12,398,070.41

Appendice B

QSOS – Definizione dei parametri delle Identity card[2]

[Atos Origin – whitepaper]

Intrinsic durability		Score		
		0	1	2
Maturity	**Age**	For instance less than 3 months	For instance between 3 months and 3 years	For instance more than 3 years
	Stability	Unstable software with numerous releases or patches generating side effects	Stabilized production release existing but old. Difficulties to stabilize development releases	Stabilized software. Releases provide bug fixes corrections but mainly new functionalities
	History, known problems	Software knows several problems which can be prohibitive	No known major problem or crisis	History of good management of crisis situations
	Fork probability, source of Forking	Software is very likely to be forked in the future	Software comes from a fork but has very few chances of being forked in the future	Software has very little chance of being forked. It does not come from a fork either

[2] QSOS – Method for Qualification and selection of Open Source Software [pag 16-27]

Intrinsic durability		Score		
		0	1	2
Adoption	**Popularity (related to: general public, niche, ...)**	Very few users identified	Detectable use on Internet (sourceforge, freshmeat, google, ...)	Numerous users, numerous references
	References	None	Few references, non critical usages	Often implemented for critical applications
	Contributing community	No community or without real activity (forum, mailing list,...)	Existing community with a notable activity	Strong community: big activity on forums, numerous contributors and advocates
	Books	No book about the software	Less than 5 books about the software are available	More than 5 books about software are available, in several languages

Intrinsic durability		Score		
		0	1	2
Development leadership	**Leading team**	1 to 2 individuals involved, not clearly identified	Between 2 and 5 independent people	More than 5 people
	Management style	Complete dictatorship	Enlightened despotism	Council of architects with identified leader (e.g: ASF, ...)

Industrialised solution		Score		
		0	1	2
Services	**Training**	No offer of training identified	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Rich offers provided by several contractors, in several languages and split into modules of gradual levels
	Support	No offer of support except via public forums and mailing lists	Offer exists but is provided by a single contractor without strong commitment quality of services	Multiple service providers with strong commitment (e.g: guaranteed resolution time)
	Consulting	No offer of consulting service	Offer exists but is restricted geographically and to one language or is provided by a single contractor	Consulting services provided by different contractors in several languages

Industrialised solution		Score		
		0	1	2
Documentation	**Documentation**	No user documentation	Documentation exists but shifted in time, is restricted to one language or is poorly detailed	Documentation always up to date, translated and possibly adapted to different target readers (end user, sysadmin, manager, ?)

Industrialised solution		Score		
		0	1	2
Quality Assurance	**Quality Assurance**	No QA process	Identifies QA process but not much formalized and with no tool	Automatic testing process included in code's life-cycle with publication of results
	Tools	No bug or feature request management tool	Standard tools provided (for instance by a hosting forge) but poorly used	Very active use of tools for roles/tasks allocation and progess monitoring

Industrialised solution		Score		
		0	1	2
Exploitability	**Ease of use, ergonomics**	Difficult to use, requires an in depth knowledge of the software functionality	Austere and very technical ergonomics	GUI including help functions and elaborated ergonomics (e.g: skins/themes management)
	Administration / Monitoring	No administrative or monitoring functionalities	Existing functionalities but incomplete and in need of improvement	Complete and easy-to-use administration and monitoring functionalities. Possible integration with external tools (e.g: via SNMP, ...)

Technical adaptability		Score		
		0	1	2
Modularity	**Modularity**	Monolithic software	Presence of high level modules allowing a first level of software adaptation	Modular conception, allowing easy adaptation of the software by selecting modules or even developing new ones

Technical adaptability		Score		
		0	1	2
By-products	**Code modification**	Everything by hand	Recompilation possible but complex without any tools or documentation	Recompilation with tools (e.g: make, ANT, ...) and documention provided
	Code extension	Any modification requires code recompilation	Architecture designed for static extension but requires recompilation	Principle of plug-in, architecture designed for dynamic extension without recompilation

Strategy		Score		
		0	**1**	**2**
License	**Permissiveness (to be weighted only if user wants to become owner of code)**	Very strict license, like GPL	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very permissive like BSD or Apache licenses
	Protection against proprietary forks	Very permissive like BSD or Apache licenses	Moderate permissive license located between both extremes (GPL and BSD), dual-licensing depending on the type of user (person, company, ...) or their activities	Very strict license, like GPL

Strategy		Score		
		0	**1**	**2**
Copyright owners	**Copyright owners**	Rights held by a few individuals or entities, making it easier to change the license	Rights held by numerous individuals owning the code in a homogeneous way, making relicensing very difficult	Rights held by a legal entity in whom the community trusts (e.g. FSF or ASF)

Strategy		Score		
		0	1	2
Modification of source code	**Modification of source code**	No practical way to propose code modifications	Tools provided to access and modify code (like CVS or SVN) but not really used to develop the software	The code modification process is well defined, exposed and respected, based on roles assignment

Strategy		Score		
		0	1	2
Roadmap	**Roadmap**	No published roadmap	Existing roadmap without planning	Versionned roadmap, with planning and measure of delays

Strategy		Score		
		0	1	2
Sponsor	**Sponsor**	Software has no sponsor, the core team is not paid	Software has an unique sponsor who might determine its strategy	Software is sponsored by industry

Strategy		Score		
		0	1	2
Strategical independence	**Strategical independence**	No detectable strategy or strong dependency on one unique actor (person, company, sponsor, ...)	Strategical vision shared with several other free and open source projects but without strong commitment from copyrights owners	Strong independence of the core team, legal entity holding rights, strong involvement in the standardization process

Services providing		Score		
		0	1	2
Maintenability	**Quality of source code**	Not very readable code or of poor quality, incoherence in coding styles	Readable code but not really commented in detail	Readable and commented code, implementing classic design patterns with a coherent and applied coding policy
	Technological dispersion	Use of numerous different languages	One main language with certain modules coded in other languages for specific and limited requirements	One unique language
	Intrinsic complexity	Very complex code requiring high level of expertise to perform modifications without generating side effects	Not very complex code but still requiring expertise in programming languages and software design	Simple code and design, easy to modify
	Technical documentation	No documentation (development guide or automatically generated doc like javadoc)	Incomplete or old documentation without conception and architectural considerations	Detailed and up to date documentation, including conception, architecture design and coding considerations

Services providing		Score		
		0	1	2
Code mastery	**Direct**	No direct expertise of the source code	Mastery of code but limited to a single person or to only a part of the source	Several individuals mastering the code and covering together the totality of the source
	Indirect	No indirect expertise on the source code	Strong mastery via external expertise provided by partners	Partnership with the copyrights owner and/or the core team

Appendice C

Sondaggio sulle metriche del software.

Per questioni di privacy non vengono indicati i dati sensibili inseriti nel questionario

How many people work in your company?

1-15	- 1	4.55%
16-50	- 1	4.55%
51-250	- 4	18.18%
251-500	- 3	13.64%
501-1000	- 9	40.91%
more than 1000	- 4	18.18%
Don\'t know	- 0	0.00%

2. What kind of applications are developed in your company?

Custom made applications	- 17	29.82%
Package of software applications	- 7	12.28%
Development of applications built on request	- 16	28.07%
Development of (parts of) applications as a subcontractor	- 7	12.28%
Development of components	- 10	17.54%

3. Are you going to develop a new application in the next:

1-3 months	- 2	10.00%
3-6 months	- 1	5.00%
6-12 months	- 6	30.00%
More than 1 year	- 2	10.00%
No	- 6	30.00%
Don't know	- 3	15.00%

4. If yes, in which area?

Comprehensive development of company processes - ERP	- 5	12.50%
Administrative management	- 4	10.00%
Management of project planning by a PDM (Product Data Management)	- 2	5.00%
Management of public relations, sales strength, and electronic business by a CRM	- 3	7.50%
Relation to suppliers, automatic reception of orders	- 1	2.50%
Production management	- 2	5.00%
Call center management	- 3	7.50%
Management of human resources	- 4	10.00%
Management of human resources	- 3	7.50%
Business Intelligence or Data Warehouse	- 4	10.00%
Other	- 9	22.50%

5. Corrective maintenance of existing applications is planned in the next:

1-3 months	- 9	47.37%
3-6 months	- 2	10.53%
6-12 months	- 6	31.58%
More than 1 year	- 1	5.26%
No	- 1	5.26%
Don't know	- 0	0.00%
Total Answers	- **19**	

6. Evolutionary maintenance of existing applications is planned in the next:

1-3 months	- 6	33.33%
3-6 months	- 2	11.11%
6-12 months	- 3	16.67%
More than 1 year	- 4	22.22%
No	- 2	11.11%
Don't know	- 1	5.56%

7. If yes, in which area?

Comprehensive development of company processes - ERP	- 9	16.07%
Administrative management	- 5	8.93%
Management of project planning by a PDM (Product Data Management)	- 3	5.36%
Management of public relations, sales strength, and electronic business by a CRM	- 3	5.36%
Relation to suppliers, automatic reception of orders	- 5	8.93%
Production management	- 3	5.36%
Call center management	- 5	8.93%
Management of human resources	- 6	10.71%
Management of human resources	- 4	7.14%
Business Intelligence or Data Warehouse	- 4	7.14%
Other	- 9	16.07%

7. Do you make decisions on purchasing and implementation new software applications?

No	- 9	52.94%
Yes	- 8	47.06%

8. Does your company use open source software?

No	- 11	52.38%
Yes, without changes	- 5	23.81%
Yes, with changes	- 5	23.81%
	- 0	0.00%

9. Which functional software measurement method do you use?

IFPUG FPA	- 16	45.71%
Mark II FPA	- 1	2.86%
COSMIC-FPP	- 6	17.14%
NESMA FPA	- 2	5.71%
Other (specify)	- 10	28.57%

10.Which software measurement method do you use?

COCOMO	- 1	1.89%
COCOMOII	- 4	7.55%
Evaluation by analogy	- 12	22.64%
Evaluation by quantitative methods (e.g. regression analysis)	- 12	22.64%
Personal Experience	- 17	32.08%
Other	- 7	13.21%
Total Answers	**- 53**	

Based on your experience, how would you rate the following functional measurement methods?

12. IFPUG FPA

0	- 0	0.00%
1	- 1	5.56%
2	- 1	5.56%
3	- 0	0.00%
4	- 1	5.56%
5	- 3	16.67%
6	- 1	5.56%
7	- 2	11.11%
8	- 3	16.67%
9	- 2	11.11%
10	- 2	11.11%
Not Used	- 2	11.11%

13. Mark II FPA

0	- 0		0.00%
1	- 1		6.25%
2	- 1		6.25%
3	- 0		0.00%
4	- 0		0.00%
5	- 0		0.00%
6	- 1		6.25%
7	- 1		6.25%
8	- 1		6.25%
9	- 0		0.00%
10	- 0		0.00%
Not Used	- 11		68.75%

14. COSMIC-FFP

0	- 0		0.00%
1	- 0		0.00%
2	- 1		5.56%
3	- 0		0.00%
4	- 0		0.00%
5	- 3		16.67%
6	- 1		5.56%
7	- 1		5.56%
8	- 3		16.67%
9	- 0		0.00%
10	- 0		0.00%
Not Used	- 9		50.00%

15. NESMA FPA

0	- 0		0.00%
1	- 0		0.00%
2	- 0		0.00%
3	- 0		0.00%
4	- 0		0.00%
5	- 0		0.00%
6	- 1		7.14%
7	- 0		0.00%
8	- 2		14.29%
9	- 1		7.14%
10	- 0		0.00%
Not Used	- 10		71.43%

16. OTHER

0	- 0		0.00%
1	- 0		0.00%
2	- 0		0.00%
3	- 0		0.00%
4	- 0		0.00%
5	- 2		12.50%
6	- 1		6.25%
7	- 1		6.25%
8	- 1		6.25%
9	- 2		12.50%
10	- 0		0.00%
Not Used	- 9		56.25%

Based on your experience, how would you rate the following measurement methods?

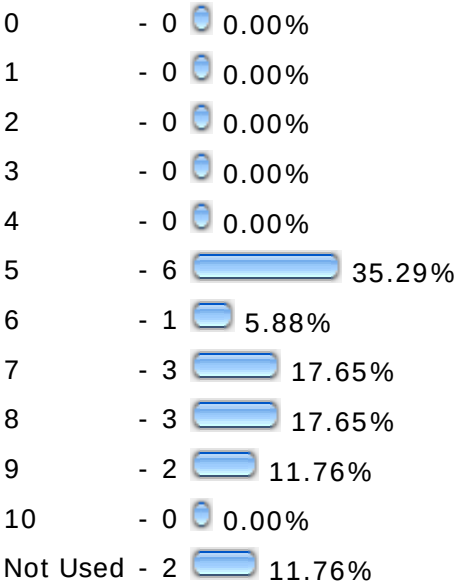
17. COCOMO

0	- 0	0.00%
1	- 1	6.25%
2	- 1	6.25%
3	- 1	6.25%
4	- 2	12.50%
5	- 0	0.00%
6	- 0	0.00%
7	- 0	0.00%
8	- 0	0.00%
9	- 0	0.00%
10	- 0	0.00%
Not Used	- 11	68.75%

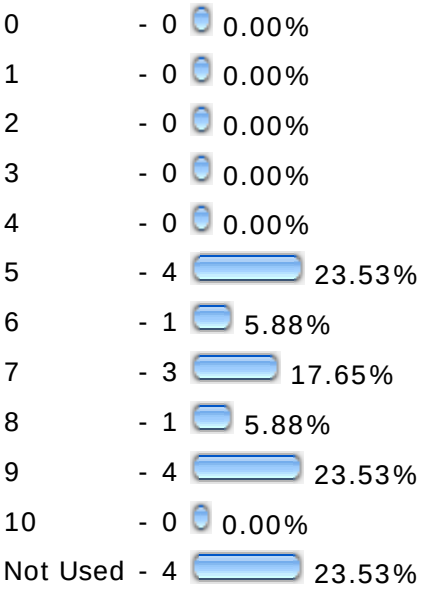
18. COCOMO II

0	- 0	0.00%
1	- 0	0.00%
2	- 0	0.00%
3	- 0	0.00%
4	- 2	11.76%
5	- 1	5.88%
6	- 1	5.88%
7	- 0	0.00%
8	- 2	11.76%
9	- 2	11.76%
10	- 0	0.00%
Not Used	- 9	52.94%

19. Evaluation by analogy



20. Evaluation by quantitative methods



21. Personal experience

0	- 0	0.00%
1	- 1	5.26%
2	- 0	0.00%
3	- 0	0.00%
4	- 1	5.26%
5	- 2	10.53%
6	- 5	26.32%
7	- 3	15.79%
8	- 4	21.05%
9	- 0	0.00%
10	- 0	0.00%
Not Used	- 3	15.79%

22. Other

0	- 0	0.00%
1	- 0	0.00%
2	- 0	0.00%
3	- 0	0.00%
4	- 0	0.00%
5	- 1	7.14%
6	- 3	21.43%
7	- 0	0.00%
8	- 0	0.00%
9	- 2	14.29%
10	- 1	7.14%
Not Used	- 7	50.00%

23. How long does it take you to estimate a software project?

2-4 hours	- 0	0.00%
4-8 hours	- 3	15.79%
1-2 days	- 8	42.11%
More than two days	- 8	42.11%

25. Which of the following metrics are being used by your company?

Lines of code	- 12	66.67%
Halstead	- 0	0.00%
Mc Cabe	- 0	0.00%
Chidamber e Kemerer	- 0	0.00%
Other	- 6	33.33%

26. What software measurement tools do you use?

- **8.** Clarity
- **7.** Self made tools (Lotus Notes X workflow and Excel for the data)
- **6.** SEER, Predictor
- **5.** In house tools and ClearQuest
- **4.** Cocomo II Construx
- **3.** Own database.
- **2.** Our own developed. Cast, HAL, Relativity and then FP Workbench, Cocomo.
- **1.** 1. Timesheet Management System 2. MS project professional 2003

Appendice D

Stima dei costi di customizzazione di sistemi ERP

La valutazione dei costi di personalizzazione del software si è proposta in modo massicio con l'avvento dei sistemi integrati di tipo Enterprise Resource Planning.

Inizialmente, i prodotti ERP fornivano la possibilità di assemblare semplicemente dei pacchetti, in base alle esigenze del cliente; in tal modo, le fasi di codifica e di test sembravano destinate a ridursi notevolmente.

Di conseguenza, il costo principale dell'adozione di un ERP in azienda era il costo di Licenza, mentre i costi di personalizzazione sembravano avere un peso inferiore.

Un azienda che sceglie di dotarsi di un ERP si dota, non solo di un sistema di supporto dei processi aziendali ma sta anche introducendo nuovi processi e andando a modificare i processi esistenti. Se la differenza tra il modello proposto dal sistema ERP e il modello aziendale è maggiore di quanto ci si aspetta, l'intero progetto ERP potrebbe non avere il successo sperato. Conseguentemente, la necessitaà di una diversificazione dei processi aziendali richiede notevoli modifiche all'ERP che comportano la necessità di aggiungere funzionalità ad-hoc attraverso lo sviluppo tradizionale.

Le componenti principale di un costo di un progetto ERP sono:

- il costo di licenza: costo dei moduli standard e dei tools di personalizzazione forniti
- il costo di customizing: costo delle risorse umane inpiegate alle modifiche dei moduli standard.

Valutazione di ERP basate sui Function Points

(A. Cavallo, M. Martellucci, F. M. Stilo, N. Lucchetti e D. Natale)

Il costo di customizing, richiede due tipi di attività: la *parametrizzazione*, per impostare i parametri di controllo di funzionalità già esistenti e funzionanti e l'*enhancement*, per sviluppare e realizzare funzionalità non previste nelle librerie delle funzionalità standard, report, interfacce e conversioni.

Obiettivo del modello è quello di capire come associare la FPA ad un progetto ERP dal punto di vista Azienda-Cliente, ricercando nel manuale IFPUG le regole candidate a misurare la novità del customizing evitando di stravolgere i contenuti di conteggio e rispettando la normativa ISO/IEC 14143-1.

Valutazione dei costi di customizing.

Una volta che l'azienda ha scelto i moduli necessari, per ognuno di questi il customer seleziona le funzionalità standard più simili alla strutura organizzativa dell'azienda e procede alla loro parametrizzazione.

Per la stima dell'effort di parametrizzazione, sono stati configurati due approcci differenti: da un lato ci sono esperti di misurazione che richiedono metodi di misurazione diversi dai Function Point, dall'altro altri esperti auspicano il riconoscimento della FPA come strumento potenzialmente in grado di misurare qualunque oggetto software compresi gli ERP.

Nel caso di parametrizzazione, il customer, per una determinata funzionalità già attiva, seleziona alcuni parametri di controllo consentiti e ne esclude altri con l'intento di alterare il comportamento del sistema introducendo le funzionalità selezionate a comportarsi come richiesto. Da queste considerazioni è possibile assimilare il processo di parametrizzazione ad un External Input (EI) quale "*processo elementare che elabora informazioni di controllo che provengono dall'esterno del confine dell'applicazione ed*

il cui intento primario è quello di alterare il comportamento del sistema", come indicato nel manuale delle regole del conteggio.

La metrica proposta dal modello introduce un nuovo elemento funzionale: l'External Input di Personalizzazione (EIP) come sesto elemento da aggiungere a quelli individuati dalla FPA tradizionale basandosi su sei elementi standard: External Input (EI), External Input di Parametrizzazione (EIP), External Output (EO), External inQuiry (EQ), Internal Logical File (ILF), External Interface File (EIF).

Valutazione dei costi di enhancement

L'enhancement è un attività di manutenzione che richiede nuove funzionalità al software ERP. Si applicano le regole del conteggio tradizionale della FPA con i classici cinque elementi.

Le funzionalità sono classificate in:

- Richieste di *funzionalità non standard*: corrispondono alle funzionalità transazionali IFPUG di tipo EO, EI o EQ.
- *Funzionalità di tipo report*: essendo solo funzionalità di interrogazione possono essere classificate come EO
- *Interfaccia*: corrispondono agli EIF, cioè alle funzionalità di tipo dati create all'esterno dell'applicazione oggetto di conteggio e da esse referenziate.
- *Conversioni*: sono funzionalità di tipo transazionali che hanno come scopo la migrazione dei dati da un archivio a un altro, possono essere classificate come EI.